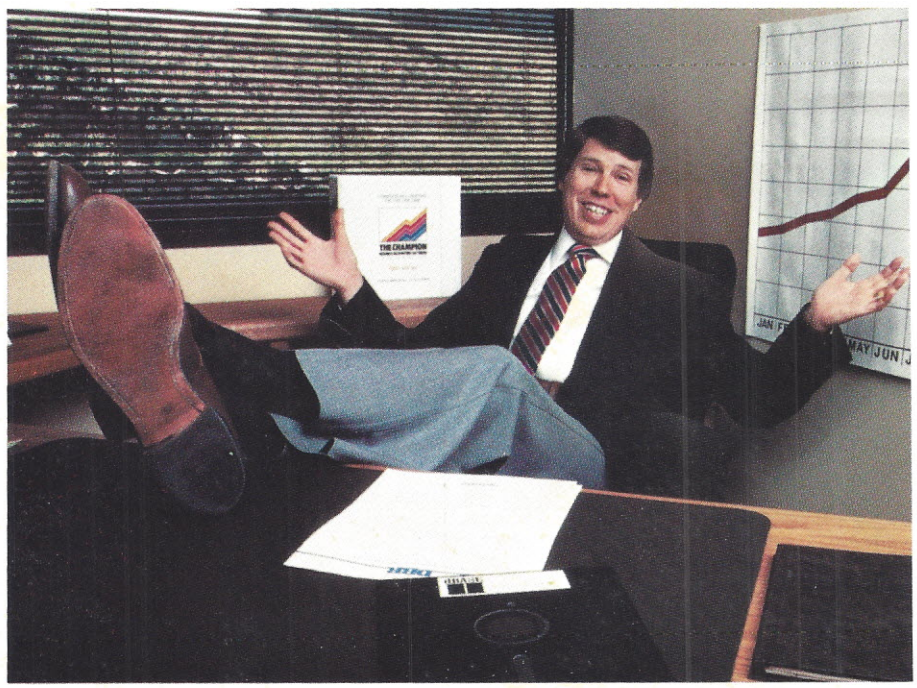
Lifelines™

The Software Magazine™

3.00 December 1983 Volume IV, No. 7 (ISSN 02479-2575, USPS 597-830)

"dBASE II® is far, far better than a shoehorn."

Rusty Fraser
President
Data Base Research Corp.



"We laughed when our customers asked us to put our minicomputer-based real-time accounting system, The Champion,™ on a micro.

"No way was it going to fit, we thought.

"We'd have to create our own database management system and, even then, it'd be a tight squeeze.

"Then we discovered dBASE II, the relational database management system for microcomputers from Ashton-Tate."

"dBASE II was a perfect fit."

"dBASE II is a program developer's dream come true. The dBASE II RunTime™ module quickly provided us with the powerful text editing, data entry speed and other 'building block' capabilities we needed to develop and deliver a new Champion to our customers—the leading real-time on-line accounting system available for a micro."

The short cut to success.

The dBASE II RunTime module has helped a lot of program developers like Data Base Research become successful software publishers.

For more about dBASE II and RunTime, contact Ashton-Tate 10150 West Jefferson Boulevard, Culver City, CA 90230, (800) 437-4329, ext. 217. In the U.K., call (0908) 568866.

For more about The Champion, call Data Base Research at (303) 987-2588.

ASHTON·TATE

Lifelines

The Software Magazine

December 1983 Volume IV, Number 7

Lifelines (ISSN 0279-2575,USPS 597-830) is published monthly at a subscription price of $24 for twelve issues, when destined for the U.S. Canada, or Mexico, $50 when destined for any other country. Second-class postage at New York, New York. POSTMASTER, please send changes of address to Lifelines Publishing Corporation, 1651 Third Avenue, New York, NY 10028.

Editorial

Better Safe Than Sorry. . .

by Edward H. Currie

A number of exciting new graphics programs have emerged for both 8- and 16-bit microcomputers. Programs such as Graftalk, Fast Graph, Cheetah, etc. now make it relatively easy to convert your data into pie, bar, line or combinations of any of the three in color or black and white. However, little is available in the form of general purpose graphics packages which provide generalized three dimensional plots. But don't dismay. There are a number of such products under preparation, one of the most exciting of which will be reviewed in a forthcoming issue of *Lifelines/The Software Magazine.* Digitizers, touch pads, light pens and mice further enhance such products and provide interesting opportunities for business and professional applications.

The graphics hardware in the S100 world is severely limited but the offerings in the 16-bit world seem to increase minute by minute. Once high resolution graphics (e.g. 1024X1024 or 2048X2048) are available for the personal computer market, sophisticated graphics and the associated software will be at hand.

Recently 16-bit hardware has begun to emerge which offers additonal screen capacity so that 40 rows by 132 columns can be viewed at one time. Windowing is also becoming an area of rapid development. Microsoft has announced their counter to Lisa and VisiON and it's reasonable to expect that an all-out war will wage as each battles to establish its approach as the standard. Then there's always the possibility that IBM will wander off in a direction of its own, leaving the traditional microcomputer software developers to fend for themselves. It's unlikely that any of these approaches will have much of a chance without IBM's enthusiastic support.

It's interesting to note that the industry has managed to bring the desk top concept to micros, which is very exciting if your desk top is approximately the size of your screen. I watched a slide presentation in which it was suggested that several documents could be viewed concurrently. The slide presented as proof positive of this showed a document, a spreadsheet and a bar chart. The only problem was that only about 10 or 15 words of the document, eight cells of the spreadsheet and one bar of the bar chart were visible at a time. It was then that I realized that the basic premise was that most of us will be content with a desk size equivalent to less than that of an eight and one half by eleven sheet of paper.

Curiously enough, the press continues to rail on about UNIX while the rest of the world tries to figure out how to use a mouse. It seems relatively obvious that this preoccupation with UNIX is largely irrelevant to most of us and, in fact, UNIX may well be forgotten on the vast majority of microcomputers. You see, MSDOS is becoming more and more UNIX-like with the addition of hierarchal file directories, pipes, and other UNIX paraphernalia. The availablity of excellent C compilers such as that produced by Lattice Corporation insure that many of the utilities supported under UNIX, which are of course written in C, can be readily moved to MSDOS. UNIX is a memory-hungry, overlay-intensive, winchester-based operating system environment when implemented on micros. Also, UNIX was designed as a development environment and has little to recommend it as a commercial environment.

In the final analysis, micros must stand on their merits and, while fancy technology is most intriguing, it has little to offer in its present form to most of us who are struggling to master the simple stuff.

AT&T is preparing to enter the market with their line of microcomputers and associated software. It is most likely that they will offer an operating system similar to MSDOS and languages which are compatible with the Microsoft products.

MSDOS for the S100 class of micros has slipped far behind, due largely to lack of interest. It's not likely that this situation will ever improve, due to the decreasing interest in S100 machines and concommitant lack of applications software. However, there have been a number of interesting developments in the 8-bit, S100 world.

The CBBS's of the world continue to make their significant contributions to the world of software, first by standardizing on the Christensen communications protocol, then by using utilities which compress files to minimize disk storage, followed by library utilities which permit a user to group families of related files in one common file. Most CBBSs are now supporting 1200 baud modems and winchester disks. Furthermore, SYSOPS have begun to place stringent time limits on a user's access time to insure that systems are not monopolized by a few. Special interest group CBBSs continue to emerge but they are a vanishingly small percentage. Database management is now an area of rapid growth as d-Base II competitors focus on the great interest in database management by micro users. One new and exciting interest in this race is a product called FORMULA II.

TI has announced their answer to IBM's Peanut. This latest member of the TI professional family is said to be fully compatible with its non-portable counterpart. The machine is provided with a nine-inch screen available optionally as either color or black and white. Memory expansion provides for up to 768K. Five expansion slots are provided. Two 5¼" floppies can be supported. Unlike the Peanut this is, in fact, a portable machine with built-in handle and storage compartments for cables. The system is graphics compatible with the Professional. Suggested retail for the portable is $2395 with one disk drive and 64K.

One interesting point about the recent demise of the TI 99/4 series is that although TI has abandoned this product line, the systems are sellouts in most retail stores and represent, for all practical purposes, the largest installed base of home machines to date. A number of book publishers were caught by surprise by TI's announcement and found themselves with books to sell on the TI series

(continued on page 17)

Feature

An Accounting Package

by Robert P. VanNatta

An accounting package

Are you looking for a moderately-priced accounting system? If so, hold off until you read about the Accounting Partner. It might be what you are looking for.

The Accounting Partner is a recently released accounting package from Star Software Systems, located at 20600 Gramercy Place, Suite 103, Torrance, CA 90501.

It is written in CB-80 and is presently available on CP/M 80, CP/M 86 and MS-DOS (including PC-DOS). The Accounting Partner comes with a very bulky three-ring binder for documentation. The CP/M-80 version that I examined was delivered on five 8-inch disks. These disks contained code only and did not contain any sample data files or the like.

The package includes a general ledger, balance forward accounts receivable, balance forward accounts payable, and payroll programs. The entire system has a retail list price of $395.

If the shoe fits, wear it!

The biggest single problem with accounting packages in my experience is finding one that "fits" the business on which it is to be implemented. The biggest single problem that I perceive with reviews of accounting packages that I have seen is their failure to provide enough information of the type that might help a reader decide if a program would "fit" or not.

The main difference is that the Accounting Partner system works.

First, it must be understood that at $395 for these four-program packages, you have essentially a low-cost system. I say that it is low-cost, not because it is cheap, but simply because it is a lot less expensive than traditionally reputable accounting packages that often list for several hundred dollars a module. My guess is that this package was put together by stripping some features out of the much more expensive "standard" Star Computer Systems modules. The result was then bunched together under one label as the Accounting Partner.

The Accounting Partner, just because it is (relatively) inexpensive, should not be confused with some of the accounting junk that was widely advertised a year or two ago. The main difference is that the Accounting Partner system works. In contemplating any computer program, the product must be looked at from a number of levels. The source code is not available for these packages, so the lowest level available for examination involves making an objective evaluation of the program's technical quality. Can the system be installed, and if so, will it perform as documented?

Installation

I found the installation routines to be quite straightforward. There was the usual menu of terminals along with a provision for a custom terminal installation. In order to get the installation to work, you must have a terminal that has a code to clear the screen and home the cursor, and a code sequence to position the cursor on the screen. Unfortunately, no facility is provided for either terminal initialization or printer initialization. I now have a couple of printers that have several different print modes which may be toggled by software from the computer. I also have a terminal that can be switched from one terminal emulation to another with an escape sequence.

It would seem to me that the availability of initialization and de-initialization on the printer and terminal ought to be standard features on all programs. The user can, of course, remember to set up his ducks before he loads the program, but what would be so bad about a data processing program that switched your multi-mode printer into the data processing mode automatically?

If you have read any of my writings before you probably already know that I tested this system on my faithful Radio Shack Model 16 with various CP/M adaptations. I was amused that Star doesn't know how to spell Pickles and Trout, and apparently doesn't know that Lifeboat has long had a version of CP/M floating around for the Radio Shack computer. They list only Pickels (sic) and Trout and Cybernetics as options. Actually Lifeboat CP/M will work under any one of about half a dozen installation options including Cybernetics. (Likewise, I found no problem using Lifeboat's SB-80).

In a moment, I am going to whimper about some of the program non-features, but first I want to go out of my way to observe that the user interface is quite good and the programs seem well written from a technical standpoint.

I am satisfied that all of the programs perform credibly as documented. The input routines appear to be well-filtered against extraneous entries (i.e., letters in number fields, etc.).

The General Ledger

The General Ledger is probably my favorite of these packages. It is organized into three logical parts. There is the chart of accounts function, the transaction entry function and the print function. The single most important thing in a general ledger is the chart of accounts. It is also traditionally one of the most difficult tasks. The problem is that the user is required to design the basic structure around which his entire accounting system is going to operate. The task of entering the chart of accounts into the program is fairly easy, but that is not the catch. The catch is that the user must understand principles of accounting sufficiently well to recognize what must be included in the

chart of accounts. If you are selling both beans and marbles, is it sufficient to have a number for "beans and marbles," or do you need to count your beans separately from the marbles?

By way of commentary, I would suggest that the design of the chart of accounts ought to be approved by the highest level of management, and probably the accountant as well.

What I am trying to say is that the chart of accounts should not be left to the computer consultant or office boy (girl?) to implement. The result is likely to be an account entitled "Beans and Depreciation," which won't do anybody any good. You also need to understand the difference between a debit and a credit if you expect to get a viable chart of accounts.

There are two basic modes of transaction entry into the General Ledger. One is the general journal and the other is the check journal. The check journal, as the name implies, permits the entry of a transaction associated with a check into the journal with an appropriate distribution to the expense areas.

Checks may be optionally written off of this journal using NEBS check forms. The general journal is for all other transactions.

All transactions are double entry, but an override is provided permitting the entry of out-of-balance transactions.

It should be noted that a big issue in general ledger programs is the allowance of out-of-balance entries. There are those who firmly believe that all such entries should be flatly prohibited and those who say that they are a necessary evil. I personally tend to think that a better design would absolutely prohibit out-of-balance in the general journal. If this is done, however, the programmer is faced with adding the additional complexity of an additional special routine somewhere else for handling the single entry postings. Accordingly, a case can clearly be made for the more flexible approach used here.

The reports are the usual ones that you would expect with a general ledger. The trial balance is particularly good in that it provides a list of all transactions organized by account number. This makes an excellent audit trail, and a convenient way to track down the inevitable problems.

The General Ledger seems sufficiently flexible that it ought to be usable for most small businesses. The program code is in the neighborhood of 175K, and, as with the other packages, can be divided among multiple floppy disks, as the chain routine will prompt a disk change if a necessary program is not found. The data files must all be on the same drive, however. The system contemplates two floppies with the programs on drive A and the data on drive B. If you follow the instructions for a hard-disk installation, however, you can map the programs to data to any drive, including the same drive (if space permits). It is hard to get a fix on just how much data space is needed, but my basic inclination is that a two-drive machine with drives of 180K each would probably prove workable for the General Ledger.

Accounts receivable

The accounts receivable portion of the Accounting Partner is a balance forward-order entry creature. It contemplates a customer list, wherein information relevant to each customer is stored. This is complemented by an invoice master record and an invoice detail record. This structure permits any number of invoices per customer and any number of items per invoice. The system also supports an "item file" in which you can place detailed descriptions of particular items. This nice feature permits the user to enter a part number and get a corresponding description on the invoice. As is characteristic of a balance forward accounting package, the payments and credit memos are posted separately from the invoices and are posted against the customer name (and balance forward) and not against particular invoices.

For those of you who haven't figured out what I am talking about, I will explain further by suggesting that accounting systems generally fall into two basic styles. One is the "open item" system and the other is the "balance forward" system.

There are some fundamental differences in the way the two types of systems work. Your monthly phone bill presents a classic example of a balance forward accounting method. Each month you get a summary of the current month's activity, and, if you haven't paid the previous month's billing, it simply shows as an unitemized "balance forward." The efficiency of such a system is built around the concept that you need only enough computer space to store the current month's detail activity and the customer list. At the end of each month or other billing cycle, a customer bill is generated out of the invoices for the month and the sum total of these charges for the month are lumped into the "balance forward." The detail file is then deleted from the computer, and the storage space recovered for the next billing cycle. Payments are simply entered by posting them against the balance forward.

One is the "open item" system and the other is the "balance forward" system.

Balance forward systems are quite popular, not because they are great, but rather, I think, because they are much easier to implement, not only in terms of software but also in terms of hardware. Contemplate for a moment the difference in storage capacity that would be required if details of each transaction were kept "on line" until the bill was paid, rather than just until it was sent.

The problem with a balance forward system is readily recognized, however, when an irate customer who has been paying $5 on his account on odd-numbered paydays for the last four months demands to know why his bill is so big. At this point the computer owner is faced with either convincing the customer that he owes that much because "the computer says so" or spending half the afternoon looking through secondary records in a vain attempt to reconstruct the account.

An "open-item" accounting system, as you might have guessed by now,

works just the other way around. Invoices are posted in the computer for each charge and they stay there until those particular invoices are paid. Payments are posted against invoice numbers, and the invoices and their associated detail information only drop out of the system when the particular invoice is paid. A customer balance will be the sum of the unpaid invoices on file. Using this sort of an accounting system, you can regenerate an itemized billing until the bill is paid or written off as a bad debt.

Your favorite $5-per-month customer can't gripe about not having an itemized bill, but, of course, he is not without gripes. The first thrust comes when the customer waves a handful of bills at you and demands to know "how cum" you sent him 10 bills for the same thing. Once you patiently explain that he got the 10th bill because he didn't pay the first nine, you get the "zonker." Your customer reaches into his pocket and pulls out a rumpled handful of $5 canceled checks, half of which are made out to your competitor across the street, and demands to know "how cum" they haven't been credited to the account.

You then have the opportunity to patiently explain that the computer only bills things that aren't paid and his $5 checks were applied against a different invoice which no longer exists because it was paid.

Which is better?

It is a truism to say that more information is better. In the real world, however, the value of the extra information must be balanced against the cost of the extra storage capacity and the administrative costs of associating payments with particular invoices. Not surprisingly, businesses with high volumes of low value merchandise tend toward balance forward methods and low volume-high value merchandise is likely to be accounted for "by the item."

The point that I am trying to make is that if your application is one in which you need (or want) an open item accounting method, the Accounting Partner won't do the job, as it is a balance-forward system.

The Accounts Receivable package contemplates a warehouse-type mercantile establishment where orders are entered, merchandise shipped, customers billed, and payments made. The system cannot be interfaced to the general ledger and is pretty limited to counting unpaid dollars. It will age the receivables and identify them by customer, but is not geared to provide information with respect to "what is selling."

It does not appear that the package would be particularly useful for non-mercantile businesses, or, for that matter, mercantile businesses which are primarily "cash and carry."

Storage requirements

I have preached to anyone who would listen, and many who wouldn't that a computer with 5.25" drives is not fit to use for general accounting purposes. The documentation with the Accounting Partner doesn't state its case just that way, but they do provide some statistical information which supports that conclusion. Specifically, they state that the Accounts Receivable system, if implemented on two single-density 8-inch drives of 241K each, will support a customer list of 100 customers who make an average of three purchases a month. Somehow, I have the feeling that that isn't even close to being adequate capacity for most applications. Likewise, I can't see even those 320K drives providing this (or any other credible accounting package) the breathing room it needs. The first accounting system that I brought up was on a pair of 596K drives, and it lasted about six months before I outgrew it. It was some years ago now, and was a different program, but the message is that it takes gobs of space to store accounting data, even for a very modest two or three person business.

Accounts payable

I have been trying for a month to think of something kind to say about the Accounts Payable program module in this package. For the most part, I am at a loss for words. My problem is not that the program is badly written in the technical sense. In fact, it appears to be almost a direct clone of the Accounts Receivable package, and therein lies the problem, in my view.

I get the distinct feeling that they came up with this package by using the global search and replace function of some editor to take the words "Accounts Receivable" out of the source code and replace them with "Accounts Payable." The problem is that you wind up with a balance forward accounts payable program.

Stated another way, the payments that you make are not associated with any particular bill or purchase order from a vendor, but rather are just associated with that vendor name.

Ideally, an accounting system ought to include a purchase order procedure wherein every bill is associated with a purchase order number.

I am a mere lawyer and not an accountant, but the thought of simply throwing money at a balance forward without being able to tell what bill you are paying or why you are paying it shocks me a bit. My concept of accounting control and audit trail requirement suggests that disbursements ought to be made only to pay identifiable bills, or purchase orders or the like, and the bill or purchase order and the check that pays it ought to be permanently matched.

To me, even a minimum accounting system should provide some method of assuring that checks only go to pay bills. Ideally, an accounting system ought to include a purchase order procedure wherein every bill is associated with a purchase order number. The purchase order numbers should be assigned in some sequence. Checks should then only be allowed to pay purchase orders by number. Ultimately, the purchase orders (and any attached bill) can be filed in sequence providing a positive record of "where the money went." This paper record should match the electronic record stored in the computer, thereby providing some semblance of the ever necessary cross-checking appropriate to accounting

systems. Needless to say, the failure of this accounts payable system to insist on positive, permanent association between payments and purchase orders makes it a program that I am not inclined to use or recommend.

This faint praise should not be construed as a statement by me that the program is no good. To the contrary, it works! It is competently programmed! It is easy to use. It performs substantially as documented. My hangup is strictly that I don't think that, in the big picture, it does what I think it ought to do. I happen to think that an accounts payable program ought to not only count your beans but also do its best to keep track of where (and why) the beans go.

Payroll

The fourth package in the Accounting Partner is the payroll package. I am rarely short of words, but when it comes to trying to talk about a payroll program I get pretty close to it. I have fixed up more than my share of payrolls over the years and they are a real pain.

The program code for these payroll routines won't fit on one 8-inch disk. I think that it is fair to say that the undertaking inherent in figuring a payroll is massive.

I would regard this payroll package as well-written and to the point. Its organization is understandable and it appears to do most of the things that a payroll program ought to do.

Its primary utility appears to be for hourly employees.

Functionally, the program will take the information from employee time cards, match it against the employee master file, and the withholding and deduction tables, and print a check for what is left over. One thing that puzzled me is that the system has a provision for deducting draws from the net pay. I could not, however, detect how you could make the draw in the first place.

As far as non-features go, the system has its share. The one that jumped out at me immediately is the complete lack of any cost accounting primitives. I suppose that I am reviewing a Moped and whining because it is not a Mercedes, but this system has no method of associating particular payroll expenses with particular jobs. I note this non-feature because it seems useful and because it is present in the Osborne Payroll program system that is in the public domain and available from CPMUG.

The fascinating question about this payroll program involves the question of defining the type of business that it would fit.

I have done payrolls involving half a dozen employees or less for years. I am unable to bring myself to believe that this payroll system (or for that matter any competitive program) is worth the bother for such a small payroll. I have learned to balance my payrolls on a spreadsheet, and for a small number of employees I am convinced that a semi-manual method using a spreadsheet is a better deal.

Obviously, there is some point where the critical mass of the payroll function justifies going to a program such as this one. I am convinced that this point lies somewhere above the six employee mark, but the exact point is probably one of personal preference.

General observations

It would probably be too much to expect that anyone were able to make use of all four of the packages included in the Accounting Partner. Considering the cost of the package, which is modest, this system is worthy of serious consideration even if only some of the modules are viable. If you can get even two of these modules to "fit," the package is likely to prove cost effective. The General Ledger is a sure fire winner for almost any small business. Finding one of the others that will "fit" is going to be a bit harder.

I have dismissed the Accounts Payable routine as being "unsafe at any speed", so the receivable and payroll routines are left. If you are in the general mercantile business and are really only interested in how much your receivables are and who they are due from, the AR will work for you. As noted, the payroll program is most useful if you have a number of hourly employees and are not concerned about generating additional management information about what they were doing.

None of these programs are interactive with each other. This means that, for example, the Accounts Receivable module does not feed information on sales and receivables to the general ledger. This non-feature adds simplicity to the programs, but greatly reduces the level of automation.

The General Ledger is a sure-fire winner for almost any small business.

Another concern that I have about this system in general is what I would call the lack of a "back door." It has been my experience that everyone underestimates the amount of computer hardware actually needed to handle an accounting task. A business, particularly if it is growing rapidly, is likely to outgrow its computer every six months to a year. The issue from the software standpoint is to have a system that can be reconfigured for the new computer or additional drives that become necessary.[1]

The data file initialization routines are a very convenient one-step process; but as I mentioned earlier, these modules contemplate two drives, one for the program module and the other for the data. If you have a computer with only two drives, and the two drives are in the 200K to 300K range, this is a reasonable mapping strategy. Accounting packages cry out for larger drives, however, and it is too bad that the drive mapping routine is not able to optimize use of high capacity drives by spreading the data among multiple drives. My favorite computers are equipped with two 1.25 Megabyte floppy drives. The best that these programs can do with this system is to locate about 200K of program code on drive A and reserve drive B for a data drive. This mapping strategy insures that about one megabyte of space on drive A is wasted. These accounting programs, like all accounting programs, need lots of disk space, and it seems sad that optimization cannot be had on a computer that is big enough to have some chance of running them. In the same vein, there are literally tens of thousands of Radio Shack Model IIs

around, often with three or four 8-inch drives. Do you suppose that the data files can be spread out over several drives? Not on your life!

Maybe the state of the art is a fixed drive, and I shouldn't be griping about weak mapping routines. The current fad seems to be a floppy in A and a fixed drive in B. If this is what you have, the mapping will work better, but your fixed drive had better be a big one if you want to use the computer for anything else. Similarly, if you had any idea of optimizing your hard drive by putting the small data files on a logical drive optimized for small files and the main data files on a logical drive configured for large files, forget that too!

The way I have it figured, if you bring up three modules of this system and have two megabytes of data associated with each module (which is conceivable), you would find the results much more satisfactory on a 2.5M floppy computer than on a 5M fixed drive (assuming the mapping routines were "wised up" enough to allow data files to be distributed to more than one logical drive).

Conclusions

I think that this system was programmed with considerable skill. The programs are fast, and the data management routines are efficient. The documentation is understandable, perhaps even to a first-time computer user. The system seems to flow naturally and I feel comfortable in saying that it is relatively easy to use. Likewise, I am satisfied that the system performs as documented.

Somehow, I feel that few people appreciate the extent of the commitment that a business makes, when a decision is made to trust the books to a computer. For the most part, it is a path of no return. Once the data-base is built, there is no going back. When it comes to the selection of accounting software, the choice had better be right the first time, because the opportunity for changeover is generally limited.

Since no company can (successfully) grow past its accounting system, it is very important to have an accounting system that can grow and change with the business needs. I happen to think that availability of source code and documentation of the data files, sufficient to permit custom modifications, is a dreadfully important issue which ought to be considered when it comes to selecting accounting systems. None of these "good things" are provided with this system. The only favorable note that I can make along these lines is to observe that since it is a CB-80 system, all of the files are in ASCII notation and, accordingly, reverse engineering on the data file contents does not appear to be too difficult.

FOOTNOTE

[1]Star Software advises that "real soon now" they will have an upwards compatible system, with more features (and a higher cost) that will accept a transfer of the data files.

U.S. Postal Service
STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION
Required by 39 U.S.C. 3685)

1A. TITLE OF PUBLICATION	1B. PUBLICATION NO.	2. DATE OF FILING
Lifelines/ The Software Magazine	0 2 7 9 2 5 7 5	10/24/83
3. FREQUENCY OF ISSUE	3A. NO. OF ISSUES PUBLISHED ANNUALLY	3B. ANNUAL SUBSCRIPTION PRICE
Monthly	12	$24 US; $50 For.

4. COMPLETE MAILING ADDRESS OF KNOWN OFFICE OF PUBLICATION *(Street, City, County, State and ZIP Code) (Not printers)*
1651 Third Avenue, New York, NY 10028

5. COMPLETE MAILING ADDRESS OF THE HEADQUARTERS OF GENERAL BUSINESS OFFICES OF THE PUBLISHER *(Not printer)*
1651 Third Avenue, New York, NY 10028

6. FULL NAMES AND COMPLETE MAILING ADDRESS OF PUBLISHER, EDITOR, AND MANAGING EDITOR *(This item MUST NOT be blank)*

PUBLISHER *(Name and Complete Mailing Address)*
Intersoft Corporation, 1651 Third Avenue, New York, NY 10028

EDITOR *(Name and Complete Mailing Address)*
Dr. Edward H. Currie, 1651 Third Avenue, New York, NY 10028

MANAGING EDITOR *(Name and Complete Mailing Address)*
Susan Sawyer, 1651 Third Avenue, New York, NY 10028

7. OWNER *(If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be given. If the publication is published by a nonprofit organization, its name and address must be stated.) (Item must be completed.)*

FULL NAME	COMPLETE MAILING ADDRESS
Intersoft Corporation	1651 Third Avenue, New York, NY 10028
Oak Management Corporation	2 Railroad Place, Westport, CT 06880
Bessemer Venture Partners	630 Fifth Avenue, New York, NY 10111
Larry B. Alkoff	1651 Third Avenue, New York, NY 10028
Anthony R. Gold	1651 Third Avenue, New York, NY 10028

8. KNOWN BONDHOLDERS, MORTGAGEES, AND OTHER SECURITY HOLDERS OWNING OR HOLDING 1 PERCENT OR MORE OF TOTAL AMOUNT OF BONDS, MORTGAGES OR OTHER SECURITIES *(If there are none, so state)*

FULL NAME	COMPLETE MAILING ADDRESS
None	

9. FOR COMPLETION BY NONPROFIT ORGANIZATIONS AUTHORIZED TO MAIL AT SPECIAL RATES *(Section 423.12 DMM only)*
The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes *(Check one)*

☐ (1) HAS NOT CHANGED DURING PRECEDING 12 MONTHS ☐ (2) HAS CHANGED DURING PRECEDING 12 MONTHS *(If changed, publisher must submit explanation of change with this statement.)*

10. EXTENT AND NATURE OF CIRCULATION	AVERAGE NO. COPIES EACH ISSUE DURING PRECEDING 12 MONTHS	ACTUAL NO. COPIES OF SINGLE ISSUE PUBLISHED NEAREST TO FILING DATE
A. TOTAL NO. COPIES *(Net Press Run)*	13,000	13,000
B. PAID CIRCULATION 1. Sales through dealers and carriers, street vendors and counter sales	1,700	1,757
2. Mail Subscription	6,000	6,117
C. TOTAL PAID CIRCULATION *(Sum of 10B1 and 10B2)*	7,700	7,974
D. FREE DISTRIBUTION BY MAIL, CARRIER OR OTHER MEANS SAMPLES, COMPLIMENTARY, AND OTHER FREE COPIES	440	310
E. TOTAL DISTRIBUTION *(Sum of C and D)*	8,140	8,284
F. COPIES NOT DISTRIBUTED 1. Office use, left over, unaccounted, spoiled after printing	4,785	4,673
2. Return from News Agents	75	43
G. TOTAL *(Sum of E, F1 and 2--should equal net press run shown in A)*	13,000	13,000

11. I certify that the statements made by me above are correct and complete

SIGNATURE AND TITLE OF EDITOR, PUBLISHER, BUSINESS MANAGER, OR OWNER
Edward H. Currie Chief Operating Officer

Answer To Last Month's Puzzle

1 B	2 L	3 O	4 C	5 H	■	6 P	7 M	8 A	9 T	10 E
11 D	I	A	R	Y	■	12 L	I	A	R	S
13 O	N	T	A	P	■	14 M	I	N	C	E	S
15 S	T	E	V	E	16 N	F	I	S	H	E	R
C	■	17 N	E	R	O	■	18 O	K	E	T	■
19 P	20 L	■	21 S	O	D	22 A	■	■	23 N	O	24 D
25 M	A	26 P	■	■	■	27 E	M	28 E	29 S	■	30 P	I
■	31 G	R	32 A	33 S	■	34 A	L	E	35 S	■	G
36 B	R	U	C	E	37 H	H	U	N	T	38 E	R
39 L	O	N	E	R	S	■	40 D	I	O	D	E
41 A	P	E	T	E	■	■	42 E	L	L	A	S
43 H	E	R	O	D	■	■	44 D	E	E	M	S

dBASE II™ made easy!

QUICKCODE™

The dBASE II Program Generator

Now dBASE II is made easy with Quickcode by Fox & Geller. QUICKCODE is a program generator, a computer program which writes computer programs.

FAST AND SIMPLE

With QUICKCODE you can generate a customer database in 5 minutes. Its that fast. All you have to do is draw your data entry form on the screen. It's that simple!

NO PROGRAMMING REQUIRED

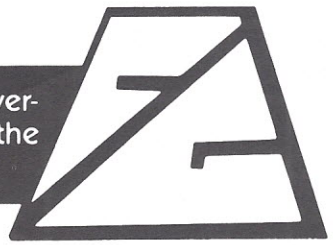
QUICKCODE writes concise programs to set up and maintain any type of database. And the wide range of programs cover everything from printing mailing labels and form letters, to programs that let you select records based on your own requirements. There are even four new data types that are not available with dBASE II alone.

YOUR CONTROL

And since you work directly with your information at your own speed and your own style, you maintain complete control. Telling your computer what to do has never been so easy.

QUICKCODE, by Fox & Geller. Absolutely the most powerful program generator you've ever seen. Definitely the easiest to use.

Ask your dealer for more information on QUICKCODE and all the other exciting new products from Fox & Geller.



FOX&GELLER

Fox & Geller, Inc. Dept. LIF 001 604 Market Street Elmwood Park, N.J. 07407 (201) 794-8883

QUICKCODE trademark of Fox & Geller, Inc
dBASE II is a trademark of Ashton-Tate

MP/M 8-16

By Bruce H. Hunter

The computer industry moves very rapidly, particularly hardware, and every so often it is necessary to update your existing hardware and software to reflect what is current in the industry. I had to do that very thing recently, and I was very pleased with the results.

For some time I'd been doing consulting work involving heavy emphasis on programming as well as writing computer books and magazine articles, and I'd managed to pick up an assortment of micro computers at various times for different purposes. Faced with upgrading our operation, I had to consider the same options everyone else has to think about today. It's not just a matter of considering whether or not to buy Brand A or Brand B anymore. Nor is it necessarily enough to narrow down the choice because of the application (i.e., word processing vs. program development). The entire computer industry is undergoing a transition from 8-bit to 16-bit machines, from single-user/single-tasking operating systems to concurrent and/or multi-user, multi-tasking operating systems, leaving a lot of people who already own perfectly good 8-bit machines right in the middle with a lot of fast decisions to make.

The bulk of the software for micros available today is for 8-bit CP/M machines, with most of it running very smoothly having had ample time to work out most of the bugs. However, 16-bit machines have the advantages of much larger available memory and an ever-expanding software base. In addition, there have been a lot of technological advances for 16-bit buss lately, including the Intel 8087 chip which gives the micro an 80-bit math word. On the other hand, if you already have owned and enjoyed an 8-bit machine, you have to think about what to do with all that software you've been buying for the last few years. The old arguments about 8-bit vs. 16-bit don't really mean much anymore. More to the point is how to find the most graceful way to survive the transition.

Here were my ideal requirements for a transition machine. So as not to give up my extensive 8-bit software library but still be able to work in 16-bit, I needed a system that would recognize 8-bit AND 16-bit software. Several users needed to be able to work in 8-bit and 16-bit simultaneously, so a multiuser system was a necessity. One very important consideration for our needs was an ample hard disk on which to store the utilities we use in program development work and writing as well as the actual programs and text on which we're working (i.e. several programming languages, a text editor, a screen editor, a data base manager and numerous other utilities for program development and text manipulation, several large programs, one entire book and half of another one, and a half dozen articles). Lastly I needed a system where all of us could access a common data base. All this for a relatively reasonable price. Now that's a lot to ask!

I ended up with a dream machine, Gifford's System 421.

Fortunately, I found a solution that answered all my needs. Up at the San Francisco Computer Faire in March I talked to Michael Gifford and Jerry Houston of Gifford Computer Systems about what I wanted in a system, and I ended up with a dream machine, Gifford's System 421. It's a multiuser system running on CompuPro 8/16 hardware with 384K of RAM and 21 meg of formatted hard disk made to run to Gifford's requirements.

What makes a System 421 (as opposed to a 321, etc.) is the amount of memory and the number of I/O ports. Gifford Computer Systems has other systems from single user to 16-user, but their System 421 best suited our needs. It has 384 kilobytes of the finest static memory you would ever want. This is the standard amount of memory allotted for four users (for our special purposes of heavy program development and heavy wordprocessing, it turned out to be marginal). Where does the 384K go? 64K is dedicated to a hard disk cache buffer and approximately 90K is used by the system. Gifford's approach to making the system responsive and giving each user the biggest part of 63K for 8-bit software is to put the system processes in high speed memory, not on disk. The advantages of this ingenious approach are well worth the 92K. The system is "right there" so there is no waiting for the system to read its way through the inevitable 2 meg of system processes, languages and whatever else is stored on 0A (0A is the storage area for the various utilities that you want to make available for everyone such as a text editor, a screen editor, a spelling program, etc.). Also the system takes up less than 1K of each individual user's memory partition. Compare that with most versions of CP/M which take 4 to 6K for the system leaving only 58 to 60K left out of the original total of 64K memory partition. 16-bit processes are something else in that the system allocates all the memory that is needed (if it's available). With a Gifford system, adding additional memory or ports is no problem, because you simply drop in any of CompuPro's memory boards from the 64K RAM 16 right up to the 256K RAM 21, and the system automatically has more memory without alterations to the operating system. Somewhere after a meg you will have to call it quits but virtual memory CPUs like the 286 will make it run like 16 meg.

All Gifford Systems are made of certified and tested boards which are burned in for an additional two

weeks to insure reliable performance as a system. Other than some minor damage inflicted by a careless shipper, the system arrived intact and working. My friend Bill Hogan, a fellow consultant and also a Gifford System owner, arrived for the unpacking, and he cabled the machine and brought it up. I was amazed as I saw all the working software from our old CCS and our old single user CompuPro brought over and mixed in with a motley assortment of 16-bit languages and utilities, a database system I had been working on, a few magazine articles in the writing and two entire books on which I was working. We ended up with a mixture of software utilities and data amalgamated into an extremely well organized and functioning supermicro system. All that is missing is a hierarchical file manager.

Hardware aside, the heart of the system is the operating system, Gifford's enhancement of MP/M-86 called MP/M-16. Developed by Gifford's engineers to meet the challenge of Bill Godbout's 8085/8088 coprocessor, this system not only runs both 8- and 16-bit software, as it also incorporates some UNIX-like features, a few of which we will examine in this article. Their system not only runs 8-bit and 16-bit software, it automatically recognizes which is which

Hardware aside, the heart of the system is the operating system, Gifford's enhancement of MP/M-86, called MP/M-16.

(the user doesn't even have to know whether he's running an 8-bit process or a 16-bit process). In addition, both 8-bit and 16-bit processes can be mixed or matched within the system.

Our system runs 16 to 20 hours a day seven days a week accommodating the early birds and night hawks in our group, which in effect means that the system has seen about three times the use of a conventional office. In spite of our heavy demand, it has shown no signs of deterioration. Only an occasional vacuuming of the fans and interior of the chassis has been required.

Increased sophistication for micros

Multiuser operating systems have been in the news a lot this past year. It's all part of what is happening to the micro. The larger and more powerful micros become, the more demands they make on their operating systems. CP/M and MP/M must behave more like OS, DOS, UNIX or any of the larger operating systems if they are to survive the expansion of the micro. Multiuser environments demand not only that computer tasks be run concurrently, but also that peripherals must be shared without conflict and files must be both accessible to privileged users and inaccessible to those that have no need to use them. If your instincts tell you that this involves a lot more sophistication and complexity, you're right! Responding to all this, Digital Research came out with their multiuser operating system called MP/M, but it is a lot more than "a multiuser version of CP/M." MP/M is more like an enhanced, upward extension of the CP/M operating system, and for those used to CP/M, MP/M is a very good way to learn about multiuser operating systems. At a glance you can see what is familiar or CP/M-like and differentiate those features from the MP/M enhancements.

There's a lot more to a multiuser operating system than having two or more terminals on your machine. Now someone has to set and reset all the things that go with these new user devices, such as passwords and privileges, operating system parameters and device allocations (such as terminals or printers). Think about a large, modern mainframe installation — it comes complete with a person in a grey, three-piece suit called a system manager. Often with a staff to assist him, his concern is the care and maintenance of the mainframe, and these duties include setting and resetting device allocations, passwords and privileges and so forth. Running a multiuser micro is not as involved as running a mainframe, but you have to be prepared to take more responsibility for the actual running of the system than you ever did under CP/M. A lot of what you heretofore took for granted is now up to you to maintain. On the other hand, you have a lot more power at your command than you ever had before. In a real sense you become your own system manager!

MP/M 8-16

Here's where Gifford's MP/M 8-16 excels. They've bridged the demands of the small owner to a larger type of operating system like MP/M by making the operating system meet the needs of a working, changing system and recognizing the necessity of the user/owner to frequently reconfigure the operating system. In other words, Gifford's MP/M 8-16 makes it much easier to accomplish "system-manager" type tasks. Most user/owners of computers are not fluent in assembly, nor are they prepared to go into the operating system and rewrite the BIOS or XIOS every time something needs changing that was not considered when the original system was written, which is what it usually takes to do the job in standard versions of MP/M. While Gifford's MP/M 8-16 has not changed Digital's basic operating system, enhancements have been made in the areas users need to modify most often such as password files, teletype (console and other devices) files, and printer files which are simple ASCII files that specify who, where and technical necessities such as baud rates, hand shaking, parity, bit width (7 or 8), number of stop bits, etc. It takes only a minute or two to add a device, reconfigure a device, add a user or change passwords. The "system manager," usually the system owner, simply brings up the required file into his favorite text editor, adds or changes the specifications, and returns the file to disk. Then the system is rebooted (not a casual operation in MP/M) to read the new parameters. *Voilá!* The change is in effect. This may seem like a trivial accomplishment to the uninitiated, but believe me, it beats trying to reconfigure an operating

system in assembly or paying a consultant to do so.

(It is interesting to note that in the past, Digital Research traditionally used assembly language as their systems language. Gifford Computer Systems uses C as their system language, and ironically right now Digital is in the process of converting over to C for their in-house systems-level work as well! G&G has used George Eberhard's CI-C86 for much of their development work, and now they are also using Digital Research's C, just recently released. You will hear about C a lot in these next few years of transition from 8-bit to 16-bit, and it will be responsible for increased portability and compatibility of languages and operating systems. Gifford's use of C to enhance MP/M-86 is just one example of many fine things to come in this industry.)

disk and user area to be erased or passed over at the operator's discretion by a simple "y" or "n" entry.

User areas in CP/M are pretty much of a curiosity. They are part of CP/M 2.2, but not commonly used. In MP/M user areas are things with which you become familiar. A typical multiuser system contains anywhere from 10 to 100+ megabytes. To use this much storage effectively requires a method of keeping track of just what is happening and where. Having 16 user areas per disk cuts the problem down to size. Don't confuse user areas with users, however. Our system is configured so that it has four terminals on which four people can work. User areas are where data are stored and manipulated. It's not uncommon for two people to be working in the same user area. Let's look at this in more detail.

small loss, however. The usual PIP commands in the fashion of CP/M 1.4 or 2.2 need one more enhancement, the ability to cross user areas. In essence a goto is needed. To show you what I'm talking about, look at the following command line:

4A>pip e:[g0]=myprog.com

'4A>' is the system prompt indicating that this user is in user area 4, drive A. The command line 'pip e:[g0]=myprog.com' takes the file in question ('myprog.com') from the current user and disk ('4A') and transfers it to drive E, user area 0 ('e:[g0]'). The use of this ability to cross user areas is particularly necessary when archiving or backing up to a floppy disk from a hard disk. Those of you that do not yet have hard disks, take heed! The day is quickly coming when all but the most trivial of installations will use hard disk as main disk storage.

The day is quickly coming when all but the most trivial of installations will use hard disk as main disk storage.

System specifics

Let's take a look at some of the commands in MP/M (and MP/M 8-16) from the standpoint of a CP/M user. Those familiar with CP/M commands will be glad to know that most of them are also in MP/M. For example, the standard directory command (DIR) exists, which lists the directory of the disk and user area. MP/M also has SDIR which lists the directory of the disk and user area in alphabetized order along with its storage size, the number of records it contains and the total usage of the disk area. It also lists system files that would normally be invisible and tells if they are read only, read/write, system or directory.

Similarly, the familiar CP/M STAT command still exists in MP/M to show status and set attributes. Gifford's MP/M 8-16, however, has added the SET and SHOW commands to set and show system attributes.

The ERA command is present in CP/M and MP/M to erase files, but ERAQ is Gifford's enhancement which erases with query. For example, a command like

eraq *.cmd

presents all the command files in the

To give the system the versatility it needs, my 21 meg formatted hard disk is divided into three logical drives (A, B, and C) which are subdivided into 16 user areas each. Here is where most of the data are stored. My system also supports a Warp drive (designated drive M) and an M-drive/H (designated drive N), each of which also has 16 user areas. These are for speedy data manipulation, but you can't store data in these "drives" because they are volatile storage. There are also two 8" floppy disk drives which are seldom used except for starting up the system, archiving and copying, and they are designated as drives D and E. This disk designation is not cast in concrete, however. The system is shipped with CP/M-80, CP/M-86 and MP/M 8-16. Usually a fourth CP/M version is present to allow the unit to operate without the hard disk in which case you designate the floppy drives as A and B. Now, whether hard disk or floppy disk, Warp drive or M-drive/H, each drive has 16 user areas which add up to a total of 112 distinct areas in which to store and/or manipulate data.

Nothing is gained without some

CP/M-86 and MP/M require the presence of a clock board to provide a system clock for interrupts to time the system functions. A side advantage of this is the ability to access the system clock. All CompuPro systems have battery backups to the system support boards where the system clock resides. The command TOD brings up the day, date and time to the second on a 24-hour clock. This, combined with a rudimentary knowledge of C or any other systems language, makes it a relatively simple task to access the system clock and get the time for your own purposes. For example, you can write your own listing utility to add the time and date to the top of all your listings so that when the smoke clears at the end of the day you know which version of the program is the latest.

The number of available system commands has grown substantially from CP/M to MP/M. Since memorizing them all is a questionable use of your precious time and working with the system manual open at all times is a nuisance, HELP commands and a HELP FILE have been implemented to give immediate aid and assistance to the user when necessary.

Tales of fantastic UNIX features have filtered down to the micro world from the world of superminis. Some of these features have been exaggerated, but many of them are true. UNIX allows the system operator or any user to ask "who" and the system will respond with who is logged into the system. Instead of the usual CCP (command console processor) to handle the user's system input, UNIX uses a Shell to take care of command line arguments. UNIX has many other nifty features like that, so you can see that it was necessary for micro operating systems to incorporate some UNIX-like qualities in order to avoid losing the top end of the micro market to the minis.

Gifford's MP/M 8-16 has done just that. One-word commands tell who is logged in and when, what the device assignments are (e.g. what printer is now actively attached to your terminal), the name of your terminal, and so on. A sophisticated mail utility will tell you at "logon" if you have mail (mail can be created and passed to any users within the system). The system operator can choose to display a "message of the day" each time you log on. Each user has a shell attached to his terminal that replaces the traditional CP/M CCP. At this time it does not have the sophisticated attributes of the UNIX shell, like a Shell command language, redirection and pipes, but who knows what the future will bring?

If you just can't wait for the UNIX-style shell, there is an interesting and viable alternative. A number of months ago I reviewed a UNIX-like utility called MicroShell for *Lifelines*. It provides the full features of the UNIX shell to CP/M including redirection, pipes and a Shell-like programming language. I thought it was a fantastic utility which greatly enhanced my rather limited CP/M capabilities. I found that I missed it very much when I came over to MP/M as there were features I had come to depend upon as old friends. I hesitated to install a CP/M utility on MP/M 8-16 because of potential conflicts, but one day I threw caution to the wind and installed it. So far, it works. I have not pushed it to the limits and I am sure some incompatibilities will emerge, but by and large it brings the UNIX Shell-like abilities to MP/M 8-16. I am told that Rick Rump, author of MicroShell, is coding an MP/M version at this writing.

The biggest difference between other MP/M versions and Gifford's MP/M 8-16 is the ability to distinguish between 8-bit and 16-bit software. MP/M 8-16 not only automatically distinguishes between 8-bit and 16-bit software, the user doesn't even need to be aware of which files are 8-bit and which are 16-bit. The way it works is by the shell checking the file directories for COM versus CMD files. If the file (more precisely the process) is 8-bit (as indicated by the COM file extension), the computer normally runs with the 8088 in control of all CPU operations. A command entered to the shell like

3C>dBase

starts the shell searching the files for the process. A well defined search path looks through the default user area and drive and finishes in user 0 of the A drive. If the search fails to find the named process as a CMD file, the search continues along the same path for a COM file. If it is found, the shell has the file read by a process called SW. SW then allocates a block of 64K and passes the execution over to the 8085 processor. The 8088 still runs the show but the 8085 runs the process. Frequently the most outstanding features of a system is not what happens but what does not happen. My first introduction to a CompuPro system was marked by the absence of long disk-access waiting time, courtesy of DMA (direct memory accessing). The Gifford version of the CompuPro system also has something absent — lack of thrashing and clumsiness while it switches from 8-bit to 16-bit and back. It is seemingly effortless.

Gifford's wrote MP/M 8-16 and CP/M 8-16 (a version of Concurrent CP/M) for their own systems. A version of MP/M 8-16 is available from and written for CompuPro's 8/16 systems, and it has many of the Gifford system's attributes. It isn't always apparent to people just getting into computing, but there is a lot of hardware dependency and interaction in an operating system. In fact, that is the very reason for its existence! As you can imagine, the workings of a system like MP/M 8-16 run far deeper than the apparent system that is viable to the user.

The significance of the hard disk and disk emulators

Most Gifford systems ship with hard disk. Hard disks provide exponential jumps in storage capacity along with substantially increased speed of access, and the system goes way out of its way to enhance the speed of its operation. MP/M 8-16 with hard disk uses a full 64K of static memory as a hard disk cache buffer. Instead of running to the disk on each and every interrupt and slowing the system down, the system takes an interrupt every 30 seconds from the clock, then it takes the data from the cache buffer and stores it to the disk. (Naturally when the system requests data from the disk, there is an immediate response.) The speed of the system must be seen and used to be appreciated because it is nothing short of phenomenal. A client of mine using a large dBASE system takes the biggest part of an hour to do a specific merging routine with the company's data base on a CompuPro 8/16 with floppy (only). Running the same program on my Gifford system using the hard disk takes a bit over seven minutes. Program development which requires extremely intensive disk accessing is cut to about an eighth of the floppy disk time.

Even more impressive is using high-speed memory to emulate disk drives. My unit is equipped with M-drive/H (designated as the N drive) as well as Warp drive (designated as the M drive). CompuPro's M-drive/H is a dynamic RAM memory board used as a high-speed disk drive with 512K of memory (dedicated disk drive emulator) whereas Warp drive is a RAM-based disk using existing static memory. By placing the compiler and linker on the "M:" or "N:" drive, putting a program together takes place with lightning speed. WordStar by MicroPro and TW (The Word, a spelling program by Oasis Systems) run much more quickly. In WordStar, for instance, the menus change before you get a chance to read the traditional "Wait" on the top line of the display (unless you are a speed reader!). Warp drive is allocated when you bring up the system — the system asks how much of the "surplus" memory you want to use for Warp drive. On the other hand, M-drive/H is always available

whether you want it or not. Many users put the operating system on M-drive/H to enhance the overall operation of the system, and a system utility is provided for just that.

load the system, and at that time you determine how much Warp drive space (if any) you wish to allocate. As the system comes up it reads a number of files, sends out the "message of the day" to the system console, reads the day's passwords, looks for any previously specified processes to be automatically run, and then allows you to do any housekeeping that needs to be done. During this time the other consoles have been prevented from logging on. You log out when you are done with your tasks, and the system goes on to the pedestrian operations of the day. As each user logs on, the message of the day is displayed, and if any mail is present for the specific user, the word "Mail" is displayed to that console.

Most of the commands are the old familiar ones CP/M users know so well which is a real plus in the initial stages of learning to use MP/M.

How hard is it to get used to MP/M? In the normal operating mode there is not a lot of difference to the individual user. Most of the commands are the old familiar ones CP/M users know so well which is a real plus in the initial stages of learning to use MP/M. Soon however, you find the presence of all those MP/M enhancements enticing, and gradually each once is tried as it catches your fancy. Instead of doing directories with the DIR command, you find yourself using the SDIR command more often, and eventually if the DIR process were erased from the system, you would never know it. Using the TOD command (time of day) becomes as automatic as looking at your watch.

Miscellaneous MP/M notes

There are many things that are flat nono's in MP/M. If we get hopelessly fouled up in CP/M, a warm boot or even a cold boot (reset) gets us out of it. Except to bring the system up first thing in the morning, booting an MP/M system will bring over a handful of very angry users threatening to shorten your life expectancy. Instead the CONTROL C (↑C) (which used to be a warm boot in CP/M) is what you use to abort a process from your terminal. You can also abort a process from any other terminal by typing "abort" followed by the process name and the terminal number it is on. Hit return and the process is gone.

Bringing the system up takes a little longer in MP/M than CP/M. The first thing you do as the system operator is turn the machine on because the hard disk must be given a few minutes to come up to speed. Then you

The "log on" procedures are controlled by the password file. It assigns specific user areas and peripherals to each user. They can be very selective, allowing certain users access to a limited number of programs or processes. For example, you can limit someone's access to WordStar only — thus, when one logs in, the system will bring WordStar up directly. If one attempts to exit WordStar, the system is exited. "Logins" and "logouts" are recorded, and utilities are present to allow dBASE (shipped along free with WordStar and SuperCalc with most Gifford systems) to sort the data for billing.

Some Gifford system hardware specifics

The hardware provided by Gifford with the system is number one quality. The Qume DSDD Floppy Disk Drives respond to DMA (direct memory addressing) and they run very fast. I am on my second set of Qumes, and I've never have had a glitch yet. The hardware enclosure for the CPU is a CompuPro with a 20-slot mother board. The power supply is just a bit smaller than the one I used to have on my Lincoln TIG welder, and every bit as dependable! You will never lose a set of boards to a faulty power supply with a CompuPro. All the boards are CompuPro certified and tested with the exception of the hard disk controller, a Konan component, which looks and works well but has one massive T025 discrete component (more than likely a power transistor) which is nearly out of the air stream at the extreme bottom left of the board. If I had never worked as a heat transfer engineer for an electronics firm it would not bother me, but I must admit that late at night I worry a lot about that board.

The hard disk, along with the power supply, is enclosed in an unmarked housing with no manufacturer's name present. It is a 27-megabyte unit (unformatted) using 5-inch platters. There is no head lock for shipping, yet it made it here to LA from San Leandro via Emery Air without incident. The first drive was shipped by a common carrier and was not so lucky. I firmly believe that the truckers ran it over in the parking lot with a loaded Kenworth just to hear the sounds it made when the wheels passed over it.

The Gifford MP/M system uses the single pair of processors rather than the usual master/slave system. The CPU runs at a very fast 8 MHz, and unless someone is compiling or doing a massive disk access, the system's performance is not noticeably deteriorated by three users. On the other hand, if two of us are compiling and the third is archiving a book chapter or article, a fourth user added on guarantees that the system will operate at a noticeably slower rate than normal, and all four users feel it. A trick I picked up is to drop the terminal baud rate from the usual 1950 baud to 1100 or less to even out the data transfer to the terminals. It appears much less "jerky."

Related to but not a part of the Gifford MP/M 8-16 system is a very fine modem system aptly called Modem 8-16. To the best of my knowledge it is the only modem system that will operate with MP/M, at least without disrupting all the users on the system. What makes Modem 8-16 different from the rest of the modem programs is its ability to use command files as well as command line directives. The command files act pretty

much the same as a SUBMIT or Shell file, but using the extensive set of commands peculiar to modem usage. CompuServe, Telex, Dow Jones or whatever is available can automatically be invoked with a single command line, files read to or written from the modem and all without "human intervention." Modem 8-16 runs with just about any modem, but it has a preference for the Ven-tel 212 and Hayes Smart Modem. If you have a dedicated line and wish to take advantage of auto answer, you have a full and automatic communications system.

You get spoiled with a system like Gifford's System 421, and although you can go upper limit all the way and buy the most expensive hardware peripherals, you have the option with Gifford of cutting some corners. My system feels like the Rolls Royce of systems to me, but it wasn't priced like one because I went for lower priced peripheral hardware wherever I could.

What makes Modem 8-16 different from the rest of the modem programs is its ability to use command files as well as command line directives.

First of all, you don't have to use expensive hardware like Ventel modems, Diablo and Qume printers, Televideo 950 terminals and the like. These are all excellent products, but I've also tried an Eagle Modem, Epson MX-100 and Okidata 82 printers, and ADDS Viewpoint and Heath H-19 terminals on my system, and all of these work just fine. The teletype or printer files need a little altering (as with any peripheral), and you have to re-cable everything, but as long as you don't get zapped by a hardware quirk, you're off and running.

Hardware hazards

Beware of the hazards of hardware quirks. For those of you out there who perceive that it would be cheaper if you mail ordered each piece of hardware separately from Joe Shlock's Computer Discount House in Bumbreeze, Iowa, and put it together yourself, don't even think of trying it unless you are a hardware wizard. The reason you buy your system from someone like Gifford Computer Systems is because THEIR hardware wizards properly configure the hardware you and Gifford's have decided upon. These cables are made to order. Gifford has a good reputation for customer service, and their techs have a good reputation for keeping hardware quirks from being a problem.

I've rarely had to use them, however, because my friend Bill Hogan is such a talented hardware wizard in his own right. Some hardware quirks are almost inevitable, but you really ask for it when you go off on your own and blindly start experimenting with random hardware by mail order. I had some problems with the Okidata printer for just that reason, but I knew the risks. The Okidata printer is only $400, and having seen it in action, I knew it was a really fast little workhorse of a printer for the money. But it wouldn't even "talk" to my system. It ran just fine on my neighbor's TRS-80, but not on my Gifford 421 no matter what I did with baud rates and stop bits. It turns out that the pinouts didn't match the computer DB-25 connector, and as so often happens with hardware quirks, a custom cable was required. Now I did not discover that myself. When it comes to hardware, I can usually hold my own pretty well, but I am more of an amateur than anything else. Thank goodness for Bill Hogan who diagnosed the problem and had it running in an hour or two with his magic soldering iron and a Hogan-made cable.

Another good source for potential hardware quirks is an ultra fancy piece of hardware with bells and whistles. The Televideo 950 is so fussy about what pins the cable DOESN'T have, even Pat Miller at Gifford was stymied for a while. Because of the nagging cable problems we had with the beast, I wouldn't recommend this terminal for everyone. However, because it has a heavy keyboard feedback that suits my non-touch-typist hands and because it is a very "smart" terminal and useful for a lot of programmatic applications, if you can get it running smoothly, it is well worth the $1000 price tag.

Generally speaking, inexpensive hardware thrives on the system. The ADDS Viewpoint terminals, one of which is on its third system, came up and ran no questions asked. My old and original Epson MX-100 is not only on its third computer, it has eaten a truckload of paper without even as much as a head replacement. It came up without even reconfiguring a single switch.

The "supermicro" in the industry today

If there is any bad news with Gifford's MP/M 8-16 system, it is that it's hell to go back to CP/M. The transition back is hard enough to give you whiplash because the commands seem so extremely limited. However, Concurrent CP/M is much more sophisticated, being able to run concurrent processes, and I am looking forward to bringing up G&G's CP/M 8-16 in the very near future.

In the meantime, here's food for thought. Gifford's heavy but not exclusive use of CompuPro boards and chassis has the advantage of delaying hardware obsolescence. Way back in 1980 Bill Godbout (CompuPro) had the foresight to create the 8085/8088 coprocessor board. Presently they are advertising no less than six front-end boards (CPU's) featuring the Z80, 8085, 8088, 8086, 8087, Intel's 80286, Motorola's 68000 (nicknamed the "68K") and National Semiconductor's 16032. On-board math processors like the 8087, 16- and 32-bit data busses like the 8086, 68000 and 16032, separate I/O processors and coprocessors are all designed to take the load off the CPU chip and therefore increase throughput. ('Throughput' is how fast you can railroad data through a computer.) Many of the boards incorporating the new processors are designed to be upward compatible. It is comforting to know that when I decide to change processors, I can do so without changing any of the other boards in the system or the hard disk or housings. Because Gifford works closely with CompuPro, if I wanted to go the 68K route to-

day, I would need little else than a 68K board cost: $512 to $820 and Gifford's operating system for same price. This is a far cry from the $10,000+ to buy a comparable 68K system.

In closing, while a large MP/M system like mine is certainly not for everybody, today a system that incorporates the feature of concurrent processes is the best choice more often than not. If you are impatiently waiting on your present machine to finish printing output, computing a spread sheet, or whatever so you can get back in and do something else, you are wasting time and money. The feature of concurrent processes is available only under a multiuser system like MP/M or a single user multiprocess system like Concurrent CP/M. Gifford's 8-16 systems come in both MP/M and Concurrent CP/M, thus giving the advantages of concurrent processes and retaining your old 8-bit software investment while allowing you to move on to 16-bit software for all your future applications and purchases. Their systems are readily updated at a minimal cost, so hardware obsolescence is significantly delayed giving you more productive years out of your investment.

I'd like to point out that if you start converting your software and hardware now, you are making "preparation meet opportunity," ensuring some protection from obsolescence. Investing in an operating system that meets present and future demands along with hardware that is capable of living up to the quality of the operating system is a guarantee that your system will still be usable in the years to come and that your software investment will continue to be protected.

Feature

Tips To Follow With T/Maker III

by Suzanne Osterlund

Simple form letter generation

Quite often it is necessary to send the same letter to a number of different people. It is especially handy to be able to do this to a mailing list which is often re-used. Here is a simple way to generate form letters and re-usable mailing lists.

This method requires two files, one containing the mailing list (MAIL.LST) and the other containing the letter (LETTER).

The mailing list looks something like this:

```
THIS IS THE FILE "MAIL.LST"
THIS IS THE FIRST TEST OF THE LETTER
The date will begin to appear on the next letter
.continue letter
King Kong
Empire State Building
New York, NY 10022
Dear Mr. Kong:
.continue letter
Joan of Arc
Arc de Triumph
Paris, FRANCE
Dear Joanie:
.continue letter
Ronald Reagan
The White House
1600 Pennsylvania Avenue
Washington, D.C.
Dear President Reagan:
.continue letter
. . .and so on
```

Notice that you could have easily had more letters with different names, and chosen which letter to send to each person on your list. To do so, you would simply edit the file, and put the name of the particular letter file in the spot after the ".continue" command. The letter or letters should be similar in form to the following example:

```
THIS IS THE FILE "LETTER"
Let me just say thank you for your tremendous response to Simpler Software. We are
happy that you have found so many uses for it in your rather unusual profession. Please
watch the press for our latest developments.
Thanks again for your support!
Sincerely,
Simon Simpler
Software Sales
.new
October 7, 1983
.
```

Notice two "tricks" in this letter. First of all, the ".new" command moves us to the top of the next letter, on which the date is printed. Second, the period in the first position makes T/Maker insert all those blank lines between where it will print the date, and where it will print the next address it gets from the mailing list. (The period will not print, as it is in effect a blank print design command. It is used solely as a space holder.) Now, when you GET the MAIL.LST file and PRINT it, you should be on your way.

Printing the labels

You can also turn this mailing list into mailing labels quite easily. First, edit the file and add a PAGESIZE print design command at the top of the file, so T/Maker knows how large your labels are. (The label used to send out the T/MUG newsletter is pagesize 9.)

Next, REPLACE ".continue letter" with ".new." Finally, DROP all lines with the word "Dear," to get rid of the salutation. If you have the word "Dear" in an address (i.e., "Dear Creek Road") you might want to instead DROP lines containing ":" to accomplish the same thing.

Your file now looks like this:

```
.PAGESIZE 9
.new
King Kong
Empire State Building
New York, NY 10022
.new
Joan of Arc
Arc de Triumph
Paris, FRANCE
. . .and so on
```

Just load up your labels, and you are ready to go.

The CP/M Users Group — Volume 92

National CP/M Users' Group
1651 Third Avenue
New York, NY 10028

The complete CPMUG™ catalog, which consists of nearly 100 volumes, is available for $10, prepaid to the United States, Canada, and Mexico; $15, prepaid to all other countries. (All checks must be in U.S. currency drawn on a U.S. bank.)

The following description is of Volume 92, the newest volume:

VOLUME 92
Release date: April 3, 1983
Description: SECRTARY word processing program.
Submitted by:
G. Young
PO Box 3218
North Hollywood, CA 91609

NO.	SIZE	CRC	NAME
			CPMUG.092
			CONTENTS OF CP/M VOL. 92
			ABSTRACT.092
			Abstract of volume contents.
			CRCK.COM
			Use to verify checksums.
			UGFORM.LIB
			CPMUG submittal form.
92. 1	4K	FD8A	ADDEND2.TXT*
			Documentation
92. 2	8K	4483	ADDENDUM.TXT*
			Documentation
92. 3	4K	6001	DEMOFILE.TXT*
			Documentation
92. 4	63K	4978	NSSEC.ASM
			DELETED (see ABSTRACT.092)
92. 5	1K	C02B	SAMPDATA.DAT
92. 6	2K	154E	SAMPLTR.TXT*
92. 7	60K	D9C8	SECRTARY.ASM
			CP/M ASM source for processor
92. 8	11K	9394	SECRTARY.COM
			Runnable version, self-configures
92. 9	4K	0361	SECRTARY.DOC
			Intro to system
92.10	2K	3BEF	USERMAN0.TXT*
			User manual, print via SECRTARY
92.11	22K	4AE2	USERMAN1.TXT*
			User manual, print via SECRTARY
92.12	19K	30A9	USERMAN2.TXT*
			User manual, print via SECRTARY
92.13	15K	736A	USERMAN3.TXT*
			User manual, print via SECRTARY
92.14	17K	235E	USERMAN4.TXT*
			User manual, print via SECRTARY
92.15	6K	0BB8	USERMAN5.TXT*
			User manual, print via SECRTARY
92.16	10K	5E7C	MERGE.BAS
			Creates SECRETARY compatible MERGE data files
92.17	22K	AE86	MERGE.COM
			Compiled version.

*Not CP/M text files. Use SECRTARY to print them. See SECRTARY.DOC.

Catalogued by Ward Christensen.

Software in the library, obtainable exclusively on diskettes, is available for a prepaid media and handling charge, as follows:

FORMAT	DESTINATION
8" IBM	U.S., Canada, Mexico—$13
8" IBM	All other destinations—$17
North Star/Apple	U.S., Canada, Mexico—$18
North Star/Apple	All other destinations—$21
KAYPRO II	Same price as Apple II.

PLEASE CLEARLY SPECIFY THE FORMAT YOU WANT WITH YOUR ORDER

This payment covers the cost of the diskette(s), packaging, and postage. Domestic shipping is via UPS where a full street address is given; all other orders are via U.S. Postal Service.

CPMUG ABSTRACT.092: Gary Young's SECRTARY word processing program.

SECRTARY is a combination editor/text formatter, using a syntax based upon Northstar BASIC. Documentation is contained in .TXT files that must be processed through SECRTARY.COM to your printer.

It is not a full screen editor, but is a nice, low-cost alternative to not having a word processor.

Things to watch out for:

- Files must fit in memory.
- Files are not compatible with other CP/M utilities (TYPE, etc) rather being in the old Processor Tech format (1 byte length, then the text, then a C/R).
- Commands are distinctly un-CP/M-like, i.e. ↑S, ↑U, and other CP/M "favorites" are ignored.
- If unspecified, drives default to A: (not to "logged in disk").
- To print the manual, the .TXT files MUST be on B:, otherwise you'll have to edit each USERMAN?.TXT and change the CHAIN B: USERMAN?.TXT at the end of each line.

04/03/83 Ward Christensen

Part 6 of the user manual refers to a program (BASICPGM), that is used to create MERGE type data files. This was not included because it is not compatible with CP/M. A program has been added (MERGE.BAS) which will allow the merge-data files to be written. Since not everyone will have the interpreter (MBASIC, a.k.a. BASIC-80, a.k.a. MS-BASIC, a.k.a. Microsoft Basic), a compiled version is included.

The merge-data files are the one exception to the above note that the files used by SECRTARY are not CP/M compatible. In fact, they may be created with most CP/M editors. SECRTARY requires that each merge-data record conform to a particular format. Records must be surrounded by double quotes, and the fields in the record must be separated by plus signs. MERGE.BAS won't allow either of these two characters to be entered into a field. It will also check for internal quotes that may have been entered with a text editor.

While LISTing or PRINTing a file SECRTARY will recognize ↑S, but only if it is the only character that is typed. The first time any other character is typed, it will be discarded, and thereafter, only the ↑S will be ignored.

Several minor bugs noted here:

1. You may LIST a file consisting of nothing but #directives, but if an attempt is made to PRINT the same file, SECRTARY will not know where to stop. It will print whatever random garbage is in memory.
2. The first time the DIR or LOAD command is used, the drive that is currently logged onto is searched. Thereafter, only drive A: is searched if the drive letter part of the filename is not typed.
3. The USERMAN?.TXT files don't always specify drive B: as part of the CHAIN command. This will eventually cause SECRTARY to abort with a "file not found" error message when PRINTing the manual. The simplest way to get around this problem (until you learn how to use SECRTARY to edit the CHAIN commands), is to use PIP to copy the USERMAN? files so that they are on both drives A: and B:.
4. One of the supplied files says that to produce a listing of the manual you should first load USERMAN1.TXT, and then PRINT it. Actually, you should first load USERMAN0.TXT. This file con-

tains the manual's index, and also the print formatting commands used for printing all of the USERMAN? files. If you start printing from USERMAN0 though, SECRTARY will abort with an error message sometime after it has CHAINed to USERMAN1. This is a genuine bug. I found two ways to get around the problem though. The first line in USERMAN1 that did not get printed is number 1320. Using Ward's disk editor DU, the last character in line 1320 was changed from a period to an exclamation mark. Problem gone. The second method found can be used by those not having DU. First load USERMAN0. Next, change the third line "1040#MARGIN 10,70" by either decreasing the first number, or by increasing the second. If you have an 80-column printer, you can type "1040#MARGIN 10, 72". If you have a 132-column printer, you may want to change the line to "1040#MARGIN 10,120". To produce a hard copy listing, now type"DEVICE L" and then type "PRINT". I don't know if the manual indicates this or not, but when the PRINT command finishes, the device is reset to C(onsole).

As mentioned in the CPMUG file, an assembly language source file was deleted from the volume. This version was the original one, and would not run under CP/M. If you are also running North Star's DOS, or are just curious, the author has sent this version to the North Star Users Group. The removal of this file allowed for the inclusion of this expanded ABSTRACT, as well as the following files:

CRCK.COM, U-G-FORM.LIB, MERGE.BAS, and MERGE.COM.

If you want to submit programs to the CPMUG, read U-G-FORM.LIB. If you want to help review submissions, check out Ward's CPMUG bulletin board. The number is (312) 849-1132. You may use any modem program to use the bulletin board. To safely transfer files though, you will have to either use one of the MODEM7?? programs available from CPMUG or SIG/M, or find a commercial program that uses the same XMODEM protocol. Two that do are ASCOM and MITE. Information about both CPMUG and SIG/M can be found on the Remote Interchange and Bulletin Board System (RIBBS) of Cranford, N.J. This system is provided as a public service by the Amateur Computer Group of New Jersey (ACGNJ) and also uses the XMODEM protocol for file transfer. The number is (201) 272-1874.

10/07/83 Bill Norris

Users Group Corner

CP/M-SIG
Cincinnati Computer Club
c/o Ric Allan
799 Converse Drive
Cincinnati, OH 45240
This group is interested in getting new members and trading info with other clubs.

PicoNet
PO Box 391566
Mountain View, CA 94039-1566
This CP/M Users' Group has been formed as a non-profit organization for the purpose of assisting in the education and training of those involved with or interested in CP/M-based systems. It also disseminates public domain software. Membership is $24 a year, which includes a subscription to *Pico News*. Monthly meetings are held in Palo Alto on the campus of Stanford University in the linear accelerator auditorium.

Utah Computer Association
c/o Aaron Christensen
(Secretary/Treasurer)
720 E. 5th Avenue, No. 14
Salt Lake City, UT 84103
General meetings of the Utah Computer Association are held on the second Thursday of each month at 7:30 p.m. at Mountain Fuel Supply in the auditorium on 180 E. 1st Street, Salt Lake City, UT. In addition to the general meetings, there are several users' groups, including the hardware group meeting and the CP/M Users' group meeting.

(continued from page 2)

which they can only sell to the installed base. Still, there may be some significant sales to be made for the next year or so into this market. It's unclear how the software suppliers for the 99/4 will react, but it is very clear that they won't be developing any new applications.

The Peanut is receiving mixed reviews but the overall response is, as of this writing, decidedly underwhelming. Rumors abound to the effect that IBM was reluctant to release this product and, to the casual observer, such reticence is not difficult to appreciate.

There are two versions of the Peanut. The minimal configuration of this machine is a cartridge-based system with 64K of RAM and sells for $699. The expanded version with one disk drive, 128K of RAM and two cartridge slots will sell for $1269.

The so-called chicklet keyboard provided has been the subject of a considerable amount of criticism. Sixty-two color-coded keys are provided and in spite of the fact that there are no function keys of the type provided for the IBM-PC, it is possible to perform all of the same functions as on the larger machine (at the expense of more keystrokes).

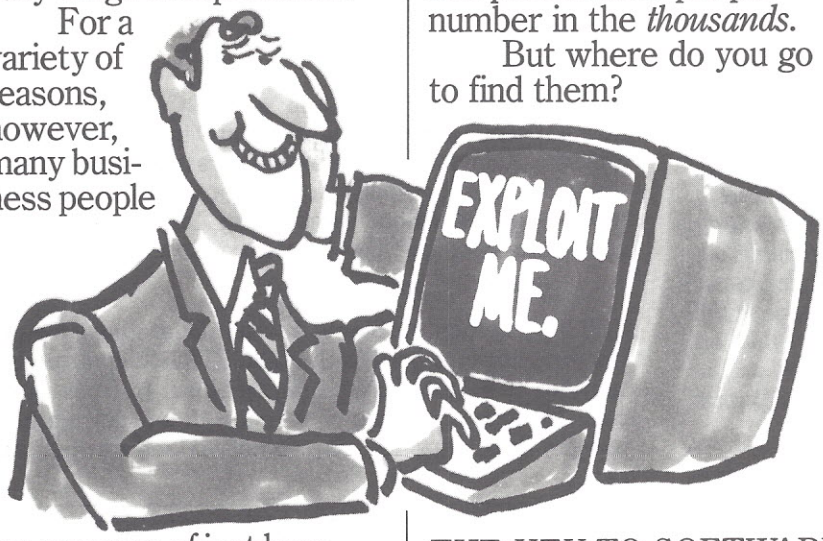
The minimal configuration supports only 40 columns, but for a small investment the full 80 columns are available. An internal, optional, Novation modem is available for $199 but it's only 300 baud. That figures, just as the rest of the world is headed for 1200 baud! The keyboard is battery powered with optional cord for noisy environments. The keyboard shuts itself off to conserve power when a user is not typing, but there could conceivably be problems if other Peanuts were in close proximity or other infrared sources were present.

In spite of the negative comments by some, the machine is interesting and undoubtedly will be of great interest to those of us with an IBM-PC who are in search of a second machine. But of course, if you are looking for state-of-the-art technology, you should look elsewhere. Just remember that, for the most part, the cutting edge of technology is more real than metaphorical. The smart ones among us go the failsafe route, but the rest of you had better lay in a good supply of Band-Aids!

YOU SPENT $4,000 ON A PERSONAL COMPUTER. FOR ANOTHER $12.50, YOU CAN GET YOUR MONEY'S WORTH.

Today's personal computers have an extraordinary range of capabilities.

For a variety of reasons, however, many business people



are unaware of just how much their computers are capable of.

As a result, they aren't realizing the full potential of their investment.

THE KEY TO GREATER PRODUCTIVITY IN A WORD: SOFTWARE.

Computers do the work. Software does the thinking.

Expanding the amount of work a personal computer can do is merely a matter, then, of gaining access to a broader array of software.

And the software programs available to business and professional people number in the *thousands.*

But where do you go to find them?

THE KEY TO SOFTWARE IN A WORD: *LIST.*

LIST is the first publication that puts software first.

It contains articles by some of the most respected names in the computer field. Written to help you get the most out of your personal computer. No matter what brand it is. No matter what you need it to do.

More importantly, *LIST* contains the *LIST Software Locator,*™ a comprehensive guide to over 3,000 personal computer programs—conveniently indexed by application, industry, operating system and hardware. You'll find detailed descriptions of applications software that pertains specifically to the type of business you're in. And the type of needs you have.

LIST is sold at leading computer stores and bookstores. Or, you can phone our toll-free number (1-800-821-7700, Ext. 1110) or send in the coupon below, and receive a copy by mail. The price, exclusive of postage and handling, is $12.50.

Which, when you think about it, is a pretty small price to pay for something that can maximize a much larger investment.

LIST *is published by Redgate Publishing Company, an affiliate of E.F. Hutton.*

I'D LIKE TO GET THE MOST OUT OF MY PERSONAL COMPUTER.

Please send me _____ copies of *LIST* at $12.50 a copy plus $2.00 each for postage and handling. (Tax will be added where applicable.)

☐ VISA ☐ MasterCard (Interbank No. _____)

Card No. _____ Exp. Date ____

Signature _____

Print Name _____

Address _____

City _____ State _____ Zip _____

Send to *LIST,* Redgate Publishing Co., 3407 Ocean Drive, Vero Beach, FL 32960.

Or phone, toll-free: **1 800 821-7700 Ext. 1110**

MasterCard VISA

LIST™

The Software Resource Book For Personal Computer Users

Feature

PL/I From The Top Down

By Bruce H. Hunter

We will look at PL/I's ability to manipulate numbers in this sixth installment of the PL/I article series, but before we do, I would like to briefly respond to the many letters I've received from Lifelines readers.

The response to the PL/I article series has been stronger than I ever dreamed it would be. Surely this indicates how many people realize the programming potential of PL/I, and their expectations of this language are not unfounded. It is my opinion that the micro implementations of PL/I are the most powerful language implementations available for general applications programming on micros today. Digital Research now has 8-bit and 16-bit versions of PL/I that run on all of their operating systems, and the 16-bit version runs on PC-DOS, the operating system of the IBM-PC. Using them in conjunction with Access Manager and Display Manager gives these compilers even greater power for applications programming.

The discerning individual who decides to learn PL/I is going to run into a few stumbling blocks, however. Digital Research's version of PL/I is a subset of PL/I Subset G, which is a subset of the full set of PL/I. Unfortunately, there are no books available on PL/I Subset G, including my own. My book was never published because I couldn't find a publisher interested in the idea. So at this time anyone interested in learning PL/I from a book on the subject must buy one on the full set and try to learn from that, all the while constantly checking to determine whether or not the material covered is implemented in Subset G (a frustrating endeavor at best).

The only source of published reading material on PL/I for micros that I know of are the manuals that come with Digital Research's PL/I compilers (the Reference Guide and Programmer's Guide). They have been rewritten, so they are quite readable now, and they offer a comprehensive look at what is available in their compiler, the only micro implementation of PL/I Subset G of which I am aware. You have to buy the PL/I compiler to get this documentation, but if you are going to use PL/I on your micro, you'll have to do that sooner or later. And their documentation is a source of reading material I highly recommend.

In the meantime, thank you for your letters, and keep them coming! Now on with Chapter Five — Number Crunching.

CHAPTER FIVE — NUMBER CRUNCHING

There are a number of qualities that mark a language's maturity. Being able to handle strings and files are both important, but a language's ability to manipulate numbers mathematically can make or break a language for many programming applications.

We tend to take a language's ability to deal with math problems for granted. Those, like myself, who came to computing by way of FORTRAN are not used to compromises! Yet a few well-accepted applications languages lack strength in this area. BASIC does quite a respectable job, but Pascal is missing several math functions, particularly transcendentals, and it has no exponentiation operator. Even the C programming language lacks polish in number handling. C also has no exponentiation operator. However, it does have power functions available, and the larger implementations of C have fine math and transcendental function libraries. C's disappointments come in the handling of statements like the following where a value is received into a variable declared floating point decimal:

```
float inpt_amt;
printf ("input amount : ");
scanf ("%f", &inpt_amt);
if (inpt_amt == 0)
  break;
```

In the expression (inpt_amt == 0), C refuses to recognize a 0 input as the same value of 0 in the constant. C's logical equal works well with integers but leaves a bit to be desired when it comes to comparing float or double. In most versions the float number inpt_amt must first be coerced to integer to make the comparison.

```
if ((int)inpt_amt == 0)
```

PL/I has no such problems. It is a fine language for number crunching, second only to FORTRAN. It strongly resembles FORTRAN in syntax and use, but has its own rules for data types and conversion. It will calculate with integer, float, exponential or binary coded decimal numbers.

We tend to take a language's ability to deal with math problems for granted.

But first things first. Let's take a look at some of the basics you need to know for crunching numbers.

Operators

Operators are what make something happen. By definition, an operator is a symbol that specifies an operation to be performed. Here is a list of the operators used for number crunching in PL/I and their corresponding order of precedence (hierarchy) and associativity which will be discussed following:

SYMBOL	OPERATOR	ASSOCIATIVITY	HIERARCHY
**	exponentiation	<---	first
↑ or ~	logical not	--->	first
+, –	unary operators	<---	first
*	multiplication	--->	second
/	division	--->	second
+	addition	--->	third
–	subtraction	--->	third

=, ↑=, <, ↑<, >, ↑>, <=, >=	relational operators	--->	fifth
&	logical and	--->	sixth
|, ! or /	logical or	--->	seventh
=	assignment operator	<---	last

(The "missing operators" in the fourth level of hierarchy are the concatenation operators used for concatenating bit strings or character strings. They will not be discussed in this number crunching article.)

Hierarchy

The order in which operations are performed within an expression is called the order of precedence. It is also known as hierarchy. You must be aware of the order of precedence when writing an expression or the results will not be what you expect. For example, the expression

```
b = c - d
```

obviously won't work if the assignment is done before the subtraction operation. The assignment is done last in order to allow the expression on the right to be fully evaluated before storing the results.

Associativity

The direction in which operations are done is important as well. All operations in an expression do not execute from left to right. The subtraction of c – d is done from left to right, but how about the assignment? In order for the results of the expression on the right (c – d) to be stored into the expression on the left (b), an assignment takes place from right to left. The direction in which operations are performed is called associativity.

Parentheses in an expression cause their contents to be acted upon first before joining the result to the remainder of the expression.

General observations

I won't be dealing with each operator separately in this article, but we will be dealing with a few by way of some comments and elementary examples.

Simple Arithmetic Operations
Simple arithmetic operations are straightforward. The expression:

```
a = b + c / d
```

is equivalent to:

```
a = b + (c / d)
```

First the variable c will be divided by d. The quotient is added to b. Lastly, the results are stored in a.

A polynomial in algebraic notation like this:

$x^2 + xy + y^2$

is written like this for use in PL/I code:

```
x**2 + x * y + y**2
```

Note that because of the order of precedence, the results are the same as if the expression were written this way:

```
x**2 + (x * y) + y**2
```

Assignment Operator
Here's another elementary example illustrating exponentiation and the assignment operator:

```
asquared = b**2 + c**2
```

In the above equation, b is squared first, c is squared next, then the results are added together and stored to asquared.

Remember that the equal sign is not always the arithmetic equal that we all grew up to love and trust. If it is acting as an assignment, the expression on the right is stored in the variable on the left. In C terminology, the variable on the left is an Lvalue.

Unary Operators
The equal sign is not the only fellow to give you a double-take. Some operators act as unary operators (what the PL/I Reference Manual calls "prefix operators"). Unary operators are those that operate on a single value. Because this is probably as clear as mud, let's look at two examples where the plus and minus signs act as unary operators:

```
Expression 1   a = b**-c
Expression 2   a = -d**3
```

Unary operators are those that operate on a single value.

In Expression 1, b is raised to the minus c, the minus being a prefix or unary operator, not a subtraction sign. Similarly in Expression 2, d is cubed first and then negated, the minus acting as a unary operator. Notice that in Expressions 1 and 2 the unary and exponentiation operators are both in the first level of precedence. You will often have expressions that have more than one operator in the same level of hierarchy. The PL/I compiler evaluates unary and exponentiation operators from right to left, while the remaining operators are evaluated from left to right. In the expression

```
a = b** -c + e + -d**3 - f
```

d is cubed and then negated and b is raised to the minus c before the addition and subtraction take place. The expression is evaluated like this:

```
a = (b** -c) + e + (-d**3) - f
```

Other operators act as unary operators, such as the logical not (i.e. "while ↑A . . ."), but we will not pursue that subject further in this article.

Comparison Operations
Let's consider relational and logical operators together. Relational and logical operators are used in comparisons to seek a Boolean true or false answer.

=	logical equal
~=	not equal
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal
!, |, \	logical or
&	logical and

Notice the logical equal is not the assignment equal. It is a "comparison" operator. The difference between the as-

signment equal and the logical equal are more easily seen in C and Pascal:

LANGUAGE	ASSIGNMENT EQUAL	LOGICAL EQUAL
PL/I	=	=
C	=	==
Pascal	:=	=

Again, the point I'm making here is that relational and logical operators have something in common — their use in comparison operations.

```
EXPRESSION 1
  if stress > 200000 then
EXPRESSION 2
  if stress >= 180000 & C < 5 then
EXPRESSION 3
  do while (stress < 200000)
```

These are always true or false conditions, true Booleans. If the expressions are true in the above examples, in Expressions 1 and 2 the ifs execute and in Expression 3 the loop continues. If the expressions are false, the ifs "fall through" and the loop exits. We will examine Boolean true and false conditions in later chapters.

Functions

Of course number crunching goes beyond the realm of the individual operators, comparison operations and straightforward mathematical equations. For instance, consider functions. PL/I has a wide variety of math-related functions in its function libraries, including roots, logs, trigonometric functions, hyperbolic functions and so on. It's a joy to number-crunch in PL/I because you have so many functions available to use as "tools."

Functions must be passed to their parameters as arguments. In the expression following, the expression in parentheses is the argument:

```
hyp = sqrt (a**2 + b**2)
```

This composite expression will square both a and b, add, then pass the result to the square root function sqrt() from which an answer will be returned. The end result will be stored in the variable hyp.

Review of numeric data types

Before moving on to program examples utilizing number manipulation, let's quickly review numeric data types in PL/I. (Data types were discussed in Installment 3).

```
Fixed Binary
Float Binary
  Single Precision
  Double Precision
Fixed Decimal
(Float Decimal not in Digital's subset of Subset G)
(Bit)
```

What the rest of the world calls integer is fixed binary in PL/I. This data type represents any whole number from −32768 to +32767. The data type fixed decimal is binary coded decimal (BCD) and will handle any number with a precision of up to 15 with both whole and/or fractional parts. Type fixed decimal is used primarily for dollars and cents calculations. Float binary is the most common for scientific work and is capable of both single precision and double precision. It has a precision of 53 with a whopping exponent of +/− 308. (Bit is a legitimate numeric data type as well but has no place in the usual arithmetic operations, at least not in the sense of number crunching, and therefore we won't be discussing it in this article.) It has been said that PL/I will do type conversion "with wild abandon." In my experience PL/I does well in type conversion but not with the ease some authors attribute to it. Type conversion is done by first converting the value to

It's a joy to number-crunch in PL/I because you have so many functions available to use as "tools."

data type character and then converting to the desired data type. Nearly a dozen conversion functions are available in the language including binary, bit, character, decimal, fixed, float, and, the most powerful of all conversion tools, the unspec function. These functions not only provide any type of conversion, they allow exact control of scale and precision as well. There is also automatic conversion, but the rules are a bit complex. Risking oversimplification, here are a couple of guidelines that are generally true:

If the operands are fixed and float the common type will be float.

If the operands are fixed binary and fixed decimal, the common type becomes fixed decimal.

The idea here is that the language will coerce the results into the data type that will risk the least damage to the accuracy of the number. It takes a language lawyer to enumerate the rules governing truncation, "min" and "max" conditions and the bounds and scope of conversion, so if in doubt use a conversion function to guarantee the integrity of the results.

Putting it to work

Now we'll play for a bit. If Bernoulli's equation following

$$\frac{p1}{w1} + \frac{(v1)^2}{2g} + z1 = \frac{p2}{w2} + \frac{(v2)^2}{2g} + z2$$

is algebraically rearranged to find pressure differential

$$\text{delta } P = \frac{(v1)^2}{2g} - \frac{(v2)^2}{2g} + z1 - z2$$

what we have is the pressure differential equal to the velocity at the origin squared minus the velocity at the exit squared, both divided by twice the gravitational constant plus elevation head at the source less elevation head at the exit. It looks like this in PL/I code:

```
deltaP = (vel1**2 / (2*g)) - (vel2**2 / (2*g)) + z1 - z2
```

Note the order of operation. The inner parenthetical expressions will be done first (two times gravity). Then the rest of the expression within the outer parentheses is performed in order of execution: first the velocity is squared, then that is divided by the 2 * gravity figure, then the sub-

traction from the first parenthetical expression by the second parenthetical expression, then height #1 is added, and finally height #2 is subtracted.

To the mechanical engineer and designer, few calculations are performed more often than spring calculations.

There is no nth root, but remembering your math, the nth root is equal to the number raised to the reciprocal of n, so

answer = number**(1/n)

By the way, for some odd reason the majority of library trig functions require radians, not degrees. This is true in BASIC as well. Sine, cos, tan, and arccos are also given in degrees. Go work for an engineering firm and give your answer in radians sometime, and your next calculations will be in the unemployment line!

All right. A little simple trig. A triangle in this corner; the near side is b and the side opposite it is a. To find the angle A

tan A = a / b

or in PL/I notation

angle_A = atand (a/b)

The same triangle for the large angle

Sin B = a/c

or

angle_B = asind (a / c)

Let's jump into a bit of engineering number crunching and calculate a few compression springs. To the mechanical engineer and designer, few calculations are performed more often than spring calculations. There are many variables in the creation of a spring — wire diameter, coil size, material, number of coils and wire tensile strength. Because of this and because the design requirements for the spring are usually restrictive, the calculation must be done repetitively to achieve an acceptable spring. It is an ideal computer application.

So let's look at a program that calculates springs in the next example. This code will look a lot different than any you've seen in this article series so far:

SPRING.PLI

VERSION 1

```
SPRING:
  PROC OPTIONS (MAIN);

  PUT LIST ('↑L                    Helical Compression Spring Program');
  RETRY:
  PUT SKIP(3) LIST ('Input Wire Dia, O D, Active Coils ');
  GET LIST (WIRE_DIA, O_D, N_COILS);
  PUT SKIP LIST ('Torsional Modulus ');
  GET SKIP LIST (G);
  PUT SKIP(2) LIST ('Specified load, ∂ Specified height ');
  GET LIST (LOAD,HEIGHT);
                                   /*  calculate spring data */
  C = (O_D - WIRE_DIA) / WIRE_DIA;
  K = (4*C - 1) / (4*C - 4) + .613 / C; /*WAHL FACTOR*/
  STRESS = (8*O_D - WIRE_DIA)*K*LOAD / (PI* WIRE_DIA**3);
  IF STRESS > 200000 THEN
CALL HI_STRESS;

  RATE = G*WIRE_DIA**4 / (8*(O_D - WIRE_DIA)**3 *N_COILS);
  FREE_LEN = LOAD/RATE + HEIGHT;
  SUM_TRAVEL = FREE_LEN - (N_COILS + 2)*WIRE_DIA;
  MAX_LOAD = SUM_TRAVEL * RATE;
  MAX_STRESS = 8*(O_D - WIRE_DIA)*K*MAX_LOAD /
          (PI*WIRE_DIA**3);
                                   /* Output Data */
  PUT SKIP(3) EDIT('Stress at working height ',STRESS) (A,F(9));
  PUT SKIP EDIT ('Total coils ',N_COILS + 2) (A,F(4,1));
  PUT SKIP EDIT
      ('Rate ',RATE,' Free Length ',FREE_LEN) (A,F(7,3),A,F(6,2));
  PUT SKIP EDIT ('Stress at closed height ',MAX_STRESS) (A,F(9,0));
  PUT SKIP(2) LIST ('Exit program or Retry  0 to exit,any no to continue');
  GET LIST (INPUT_NO);
  IF INPUT_NO = 0 THEN
      STOP;
  ELSE
      GOTO RETRY;

  HI_STRESS:
  PROC;

  PUT SKIP EDIT ('Stress = ',STRESS,' PSI') (A,F(9),A);
  PUT SKIP LIST ('Continue 0 or Restart 1    ');
  GET LIST (INPUT_NO);
  IF INPUT_NO = 1 THEN
      GOTO RETRY;
  ELSE
      RETURN;
  END HI_STRESS;

  DCL
      PI FLOAT  INIT(3.14157)  STATIC,
      INPUT_NO FIXED(1)  STATIC,
      (G,STRESS,MAX_STRESS)  FLOAT  STATIC,
      (WIRE_DIA,O_D,N_COILS,LOAD,HEIGHT)  FLOAT  STATIC,
      (C,K,RATE,FREE_LEN,SUM_TRAVEL,MAX_LOAD) FLOAT
          STATIC;

END SPRING;
```

Before getting into the contents of the code, notice a few deviations from our usual program examples. First of all the code is in uppercase. Also the code is "square." There are no procedures outside of the main except for HI_STRESS. This program just grinds its way down from the top to the bottom in a FORTRAN-like manner. Also, the declarations are at the bottom. Now there is nothing in any of these techniques that will keep the compiler from compiling this code. The point I'm making is that there is no one "right" way to program. After the basic rules of a language are learned, the rest is up to you — you should feel free to develop your own programming style. It makes programming less intimidating and more personally creative. More often than not there are several good ways to code a program, and as long as your code is clearly written and easy for someone else to follow, there are no other limits other than the compiler's. As a matter of fact, the above program is in line with the top-down philosophy of coding where the main algorithm comes first after which the the remainder of the program is developed. The algorithms of scientific programs tend to be more straightforward, and they often lend themselves to this type of format. This code is far from perfect, however, and some of it is going to come under attack during analysis.

►

Starting at the bottom, notice everything has been declared float except an input integer variable INPUT_NO. The precision falls to the default precision of binary 24 or 2 to the 24th (16777216). By keeping to a single data type, the expressions can be processed without coercion, the process of converting from one data type to the other. If N, the number of coils, were declared as fixed, for example, it would have to be coerced to type float before it could be processed.

Going to the top, the values of the input variables are input with or without decimals. The number of coils can be "10" or "10." or "10.0." PL/I does not have to be held by the hand like FORTRAN.

The coil to wire diameter ratio C is calculated by dividing the mean diameter of the coil by the wire diameter:

```
C = (O_D - WIRE_DIA) / WIRE_DIA
```

WIRE_DIA is first subtracted from O_D to give the mean (average). This is then divided by WIRE_DIA and stored as a floating binary number in the address of C.

C is now used to calculate the Wahl factor K:

```
K = (4 * C - 1) / (4 * C - 4) + .617 / C
```

In the two parenthetical expressions, C is multiplied by 4 before the subtraction takes place. The division of the two results occurs, and then the constant .617 is divided by C. Lastly, the addition takes place and the result is stored in K. In other words, the expression is evaluated this way:

```
K = ((4 * C - 1) / (4 * C - 4)) + (.617 / C)
```

After calculating stress comes a test. If stress is greater than 200,000 psi tested by the relational operator ">" (.GT. in FORTRAN), a call is invoked to the labeled procedure HI_STRESS.

Spring rate is calculated as

```
RATE = G*WIRE_DIA**4 /
    (8*(O_D-WIRE_DIA)**3*N_COILS);
```

Because of the parentheses, the mean diameter (O_D - WIRE_DIA) is calculated first. The exponentiation is done next, and then the rest of the operations are performed. The segment of the code where the calculations are done is straightforward and relatively easy to follow.

What needs some scrutiny on our part is the section of the code where results are output. If we were to simply output the results with PUT LIST, the numbers would come out as semi-nonsense. A free length could be output as 3.7539517E+02 which is a hard way to say it is 37 and 9/16 inch long. Using the format F(6,2) it will now be expressed as 37.54, a perfectly reasonable figure. The edited output statement skips three lines, outputs the string 'Stress at working height ' along with the value of the variable STRESS as a 9-digit number or less with no decimal places:

```
PUT SKIP (3) EDIT ('stress at working height ',
    STRESS)
                    (A,F(9))
```

Skipping down to the if statement, we see a branch to the RETRY label, and again in the procedure HI_STRESS another GOTO RETRY label. This is an abominable way to avoid a loop. One method of improving this code would be to trap it within a "do-forever" loop to avoid the labeled backwards goto. Try rewriting it as practice.

The point I'm making is that there is no one "right" way to program.

Let's consider coercion a moment. Imagine that in a moment of madness we had declared N_COILS fixed. Going to the expression

```
SUM_TRAVEL = FREE_LEN - (N_COILS + 2) *
    WIRE_DIA
```

PL/I would pull the variable N_COIL up to type float. We can also convert by function so as to leave no doubt as to our intentions:

```
DCL
    WIRE_DIA FLOAT,
    W_D FIXED;

.
.
WIRE_DIA = FLOAT (W_D);
SUM_TRAVEL = FREE_LEN - (N_COILS = 2)
  * WIRE_DIA;
```

That's coercion. The function float converts W_D, a fixed binary number, to type float binary.

Now let's look at this program in "my" PL/I style, with heavy emphasis on clarity and structure. First of all, everything has been declared double which eliminates conversion problems. The biggest change is that the code has been broken up into a series of small procedures — now the only job of the main procedure is calling the other procedures. In addition the main has been enclosed within a do-while loop to allow the program to iterate until the user chooses to exit the program. Also, all labeled gotos have been eliminated by altering the logic to allow the program structure to give automatic returns. Take note of the %replace statements used to create constants for PI and TRUE and also to define the CLEAR string. With all of these changes, you should notice this coding style makes the program easier to read and understand:

spring.pli
version 2

```
spring:
  proc options (main);

  dcl
    input_no      fixed(1),
    (stress, max_stress)      float(52),
    (g, wire_dia, o_d, n_coils, load, height) float(52),
    (c, k, rate, free_len, sum_travel, max_load) float(52);

  % replace
    PI by 3.1415927,
    TRUE by '1'b,
    CLEAR by '↑[:';      /* TeleVideo 950 */
  do while (TRUE);
    put edit (CLEAR) (a);
    put skip (4);
    put list ('          helical compression spring program');
    call input ();
    call calc_spring ();
    call output ();
  end; /* do */
  input:
    proc;
```

```
      put skip(3) list ('input wire dia, o d, active coils ');
      get list (wire_dia, o_d, n_coils);
      put skip list ('torsional modulus ');
      get skip list (g);
      put skip(2) list ('specified load, ∂ specified height ');
      get list (load,height);
   end input;

calc_spring:
   proc;
      c = (o_d - wire_dia) / wire_dia;
      k = (4*c - 1) / (4*c - 4) + .613 / c; /*Wahl factor*/
      stress = (8*o_d - wire_dia)*k*load / (PI* wire_dia**3);
      if stress > 200000 then
         do;
         put skip edit ('stress = ',stress,' psi') (a,f(9),a);
         put skip list ('enter 1 to continue 0 to restart ');
         get list (input_no);
         if input_no = 0 then
            return;
            end; /* do */
         rate = g*wire_dia**4 / (8*(o_d - wire_dia)**3 *n_coils);
         free_len = load/rate + height;
         sum_travel = free_len - (n_coils + 2)*wire_dia;
         max_load = sum_travel * rate;
         max_stress = 8*(o_d - wire_dia)*k*max_load
/(PI*wire_dia**3);
      end calc_spring;

   output:
      proc;
         put skip(3) edit('stress at working height ',stress) (a,f(9));
         put skip edit ('total coils ',n_coils + 2) (a,f(4,1));
         put skip edit
         ('rate ',rate,' free length ',free_len) (a,f(7,3),a,f(6,2));
         put skip edit ('stress at closed height ',max_stress) (a,f(9,0));
         put skip(2) list
            ('exit program or retry 0 to exit,any no to continue');
         get list (input_no);
         if input_no = 0 then
            stop;
      end output;
   end spring;
/**/
```

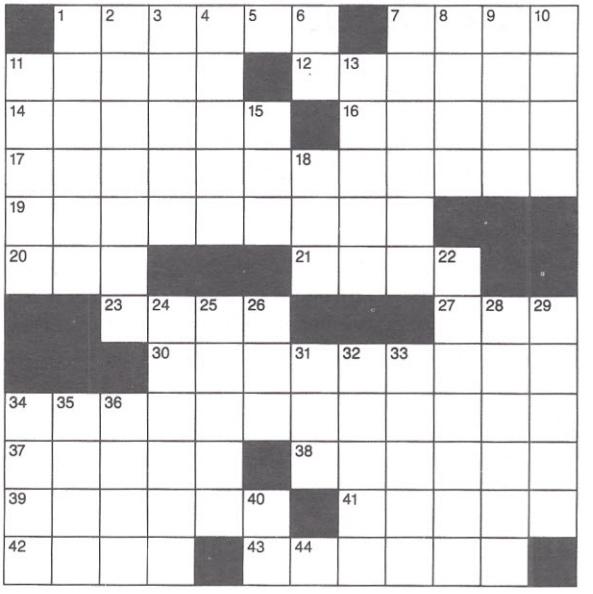
Closing remarks

Naturally this is only a beginning glance at PL/I's number crunching capability. There are so many other possibilites to consider. In C, BASIC, and Pascal utilizing matrices for determinants, finding complex mathematical solutions by way of Gauss' method of elimination, using the wonders of the Taylor series and creating your own transcendentals using Chebyshev polynomials would be very difficult indeed. FORTRAN and PL/I have both the function libraries and array capabilities for handling this order of mathematics. If you put PL/I through its mathematical paces you will find it capable of doing anything this side of complex numbers and large scale array manipulation (FORTRAN ANSI 77). The release of DRI's PL/I-86 for the 8086 and 8088 with support for the 8087 math processor gives the micro close to the power of a main frame. The 8-bit version PL/I-80 v1.4 emulates the 8086 version and gives close to the same power, sacrificing only speed.

In the next issue I will go into the heavier side of PL/I, based storage (like the heap) where storage can be allocated dynamically. We will take a look at pointers, a tool that allows access to anything, even the operating system. Also we will be getting into data structures, a fascinating method of data storage that builds in pointers to related data. ■

Crossword Puzzle

by Crescent Varrone edited by Rosalee Feibish



Across

1. Structural Design Language; from Apple?
7. Math integer librarian.
11. Enticer.
12. Needs a rest: has a __.
14. Journey account, with "ary."
16. Type of dot.
17. *Lifelines* editor.
19. Bishop's post.
20. Dry. (Fr.)
21. *Raison d'__.*
23. Stepped.
27. ". . .devious enough to be __ious?" (video arcade game).
30. Greek philosopher.
34. *Lifelines* author.
37. Spanish warning.
38. Mellow red wines of France.
39. Sacred city of Saudi Arabia.
41. Seven. (It.)
42. __ mode, as opposed to Edit mode.
43. Scandinavian city.

Down

1. Stitch. (med.)
2. Cut into three equal parts.
3. Pertaining to the kidneys.
4. Plant species substituted for flax in India.
5. With 44 Down, buttons on an elevator.
6. __ function: natural log.
7. Slogan for eighth-letter fans?
8. Lazily.
9. Skywalker.
10. Regan's father.
11. Uses a high-level, non-numeric programming language.
13. Separated.
15. Steep or macerate.
18. Prosecute.
22. Method of leaving loop structure (not "INT").
26. __ *in the Sun,* Lorraine Hansberry drama.
25. A Bean.
26. __ Eq. (topic in higher math).
28. __ical Engineering.
29. A form in which less prosaic literature is written.
31. SOE.
32. After Mond.?
33. __ times; yore.
34. Descriptor Attribute Matrices.
35. With. (Fr.)
36. __ Blue, 1971 Cy Young Award winner.
40. Eleventh month of the Jewish calendar.
44. See 5 Down.

Mea Culpa

Software Notes

by Bruce H. Hunter

I have been pleasantly surprised by the response to the PL/I articles series, and I'm pleased that so many of you enjoy it. The PL/I book has never been published for lack of a publisher willing to take a chance on PL/I as a "popular" language, so the book has never been thoroughly edited. Although I try very hard to catch every mistake, some errors slip by, and a few astute readers have found some of them which I want to pass along for your information:

Installment #1 (June issue):
On page 23 an extra space after the word "hot" and before the word "dog" were erroneously put in, as follows:

```
put list ('Hot ');
put list (' dog');
```

As this reader pointed out, the put list statement always separates each entry with a space, so this would print out with three spaces between "hot" and "dog," like this:

```
hot   dog
```

Of course that's not what you want. The put list statements should have looked like this:

```
put list ('Hot');
put list ('dog');
```

Installment #2 (August issue):
This same reader noted that a similar error occurred in Installment #2, page 12, where the statement

```
put skip list ('The amount is $', amount);
```

will be printed out with a space between the dollar sign and the number, like this:

```
$ 100.00
```

To be absolutely sure that your output will look precisely the way you want, I suggest using the picture attribute of edited output. That way you can be sure that there will be no elusive extra spaces lurking about.

On page 14, it was pointed out that the Guest_list program fragment had a semi-colon after the procedure "print_invitation," and it should have been a colon. In addition, the variable "i" was not declared. This reader also noted that the Guest_list program will print an invitation for Uncle Harry (horror of horrors!). Although the program does stop after it tests for Uncle Harry, his invitation is the last one printed. This reader created an Uncle Harry program "as a learning exercise" which follows:

```
guest_list: /* Sample program from Lifelines, August 1983
               Revised 9/19/83 WPM */

  proc options(main);
    %replace CLEAR by '↑Z';
    dcl
      name char(128) var,
      i fixed binary;
    put list (CLEAR);
    do i = 1 to 100 while (name ↑= 'Uncle Harry');
      put list ('Enter the name of the guest ');
      get edit (name) (a);
      get skip;
      if (name ↑= 'Uncle Harry') then
        call print_invitation(name);
      end; /* do while */
      print_invitation:
        proc(name);
        dcl
          name char (128) var;
        put list (name,'is invited.');
        put skip;
      end print_invitation;
  end guest_list;
```

Notice that he changed the program to send output to the console.

On page 15, in the section on exception processing, I used a fragment closing a file within the begin-end block of an 'on endfile.' A reader aptly pointed out that in Version 1.3 you may not close a file within an ON-unit. The program flow must be directed outside of the ON-unit before closing the file. You can work around this by calling a procedure to close the file, however.

On page 16, I talked about labels, and I used the example

```
call gucci;
```

This will work, but it is not good practice. PL/I theoretically requires all calls (procedure and function) to have a set of parentheses, and if there are no parameters to be passed, the call should have an empty set of parentheses following, like this:

```
call gucci();
```

For some reason, the PL/I compiler lets you get away with no parentheses sometimes, but it is much better practice to get into the habit of using parentheses because in other languages, such as C, a call without parentheses (empty or otherwise) won't even compile — you will get an "undeclared variable" error.

On page 16, illustrating returns, there was one return that didn't belong since it has no parameter to return. It is of course possible to do multiple returns from a function, but they cannot be mixed returns. The following trivial program gives a better example of returns:

```
x1:
  procedure options (main);

  dcl weight fixed,
    string char (32) var;

  %replace
    TRUE by '1'b;

    do while (TRUE);
      put skip list ('enter weight : ');
      get list (weight);
      string = opinion (weight);
      put skip list (string);
```

```
end; /*do*/

opinion:
  proc (w) returns (char (32));

dcl w fixed;

if weight = 0 then
  return ('what ?');
  else if weight > 200 then
    return ('fat');
    else if weight < 100 then
      return ('small');
      else
        return ('normal');
  end opinion;
end x1;
```

I want to thank Marc E. Cohen of North Country Computer Services in New Hampshire and William Meacham of Austin, Texas for pointing out these corrections. I appreciate all of your letters so much, and I want to thank all of the people who have been kind enough to take the time to write. The IBM-PC has increased the popularity of 16-bit machines, and this is very good news for programmers. Although there are many 16-bit language implementations available, most language compilers do not make efficient use of all that increased memory yet. To the best of my knowledge, the only language and version to date that is capable of making truly efficient use of 16-bit memory is Digital Research's C. But future releases of PL/I will undoubtedly be able to incorporate more and more of Subset G, including having PL/I-86 able to select memory model size like DR C does now. This means that PL/I-86 will be able to use all of memory efficiently like a true 16-bit language, only the beginning of bigger and better things to come.

Feature

CB80 Technical Forum

by John S. Coggeshall

The growing CBASIC Compiler family, by Digital Research, is one of the most flexible and powerful general-purpose languages available today. In this and future articles, I plan to discuss CB80, the compiler for the Intel 8080 microprocessor. My emphasis will be fairly technical; I seem to be incurably addicted to tuning (STRUCTURED) programs so as to get the most out of a language and a chip. That addiction is fed by the sublime satisfaction of inventing a beautiful solution, but it does now and then get in my way. Still, it has led to some useful discoveries, which I would like to swap with you as we go along.

Ed Yourdon has been quoted ([3], p. 313) as saying, "It is easier to make a working system efficient than to make an efficient system work." My assumption will be that a program already works and that one can possibly tighten it up without stripping the threads.

The central focus, then, will be on ways to take advantage of, and sometimes to outsmart, certain features of CB80, documented and otherwise. Each installment will present a main topic or two and perhaps some odds and ends. I'll be in touch with DRI so as to keep things in perspective and up to date, and to avoid being burned at the stake for heresy. I solicit your participation; please contact either *Lifelines/The Software Magazine* or me: John S. Coggeshall, 151 West School House Lane, Philadelphia, PA 19144, (215) 849-1521.

Odds and ends

Monte T. Schmiege wrote in response to my July article with a couple of ideas and corrections. First the corrections, to the listing on p. 31:

The double hyphen in the assignment line with CHR$(21H) is hard to see.

BDOS% = SADD(BDOS$) + 1 should read:
BDOSE% = SADD(BDOS$) + 1.
FNBDOS% + BDARG% (near the end) should, of course, read FNBDOS% = BDARG%.

(Another typo concerns FNPOKE2%: call it what you will, but there should be only one user-function in the article with 'POKE' in its name.) ►

Mr. Schmiege had gone through the same process as I had in converting to CB80. One of his suggestions, for implementing a CBASIC-style CALL in CB80, was already in the pipeline as a "Tip and Technique" (Function XFER$) in September. His other idea was for implementing my CBASIC version of FNBDOS% in CB80:

first, replace it with

```
DEF FNBDOS%(FCODE%,DEARG%) EXTERNAL
    FEND
```

The central focus, then, will be on ways to take advantage of, and sometimes to outsmart, certain features of CB80, documented or otherwise.

The calling syntax is the same as for my version — except, of course that, you can also use CALL if you don't need the result. Now code the following in assembler and stick it in your relocatable library:

```
PUBLIC   FNBDOS          ; Six-char name, linker
                         ; can't see '%'
FNBDOS:  POP    HL       ; Return Point
         POP    DE       ; BDOS Argument
         POP    BC       ; Function Code
         PUSH   HL       ; Restore Rtn Pt, fix stack
         JP     5        ; Return through BDOS
```

Flying integers

DRI has gone to some trouble to handle integers efficiently in CB80. You will find delightful examples of compiler intelligence directed to that end, as well as tantalizing cases where the code is the bulkier but faster choice. An example of the latter is ordinary subtraction: the compiler generates twelve bytes of code that uses 50 clock cycles instead of trimming three bytes with a call that would cost 27 more cycles. (Here, and in most cases below, I count only the instructions needed to get the arguments and leave the result in the HL register pair. This way, the discussion applies to an expression standing alone as well as to one that is a part of a larger expression. Also, I count a call as 27 cycles: 17 for the CALL instruction plus ten for the RETurn.)

I said "ordinary" subtraction; that is, subtraction of an integer constant greater than three, or of a variable. But if you subtract three or less, CB80 shows off by simply decrementing the value; the expression I%-2 generates only five bytes of code that executes in 28 cycles:

```
LD    HL,(I%)
DEC   HL
DEC   HL
```

You can force CB80 to "extend" this by subtracting two or more small constants in succession. For example, you can write I%-5 as I%-2-3 to get five decrements — saving four bytes and four cycles. Obviously, this sort of thing can get out of hand; I would not recommend it except in highly repetitive inner loops. But CB80 gives you a choice.

CB80 is intelligent with addition as well, although the payoff is less dramatic. The expression I%+k generates (3+k) bytes, for (16+6k) cycles:

```
LD    HL,(I%)
INC   HL     (k times),
```

if k<=3; otherwise we get seven bytes and 37 cycles:

```
LD    HL,(I%)
LD    DE,k
ADD   HL,DE
```

In this case, there is nothing to gain by forcing more than three increments. Note that the order makes a difference; k+I% takes no short-cuts regardless of the size of k. As VanNatta ([1], p. 5) mentioned in a slightly different context, put the variable first.

Because ordinary addition is tighter than ordinary subtraction, you can save five bytes and 13 cycles by coding subtraction of a constant as addition of its negative: I%+(-123) generates

```
LD    HL,(I%)
LD    DE,65413        ;65536-k
ADD   HL,DE.
```

The expressions I%+ -123, I%+ -7Bh, and -7Bh+I% all produce exactly the same code.

On the subject of representing integer constants, note that a number outside the famous range of plus or minus 32K will compile as a real number unless it is in hexadecimal notation. The assignment I%=40000 generates

```
LD    HL,pointer.to. real.constant
CALL  ?CIHM           Convert Memory to
                        Integer in HL
LD    (I%),HL
```

whereas the equivalent I%=9C40h (or I%=-25536) becomes

```
LD    HL,40000
LD    (I%),HL
```

— saving three bytes of code plus eight of data as well as (27+?) cycles. Of course, the assumption here is that you are going to use it as an unsigned integer, such as an address.

If you like the way CB80 adds and subtracts small integers, you'll love its treatment of multiplication by small powers of two. The expression k*I%, where k=2↑p, and k<=8, generates (3+p) bytes, for (16+11p) cycles:

```
LD    HL,(I%)
ADD   HL,HL           (p times)
```

— CB80 simply doubles the value! Otherwise, it multiplies the hard way, at nine bytes and (53+?) cycles:

```
LD    HL,(I%)
LD    DE,k
CALL  ?MIDH.
```

Here again, the order matters, but this time put the constant first; I%*2 triggers the library call. And again, you can force more than three doublings with expressions like 4*(8*I%). (Without the parentheses this codes as 32*I%, so keep them.) This pays off for fairly large powers of two; the multiplication routine probably takes a good 2000 cycles. Even such abominable contortions as 2*(4*I%+I%), to

multiply by ten, can pay off in speed if not in code size (that particular horror generates only eleven bytes, for 80 cycles). By the way, as you may well have conjectured by now, the expression I%+I% generates the same code as 2*I%; there is no longer any reason to avoid multiplication by adding.

Time out for a short speech on program structure and maintainability. Use obscure expressions like the above with circumspection, and only where: a) speed is very important, b) there is little chance you will ever want to change the effective constant, c) it is well commented, and preferably d) it is buried inside a tight and much-used library routine which you hope never to look at again, proud though you are of its elegance. Choice carries responsibility. Let us pause here for a moment of silent contemplation.

If you like the way CB80 adds and subtracts small integers, you'll love its treatment of multiplication by small powers of two.

Jogging integers

Oddly enough, CB80 seems to perform no clever stunts with integer division. We might have expected by now that the expression I%/k, where k=2↑p, would generate the code for SHIFT(I%,p). No such luck; you will have to attend to such details yourself. Obviously, shifting powers of two is far faster than division, and they both generate the same amount of code. Similarly, I've found no tricks in the coding of the power operator. The routine itself handles integer powers by successive multiplication (as contrasted with the use of logarithms in the case of real numbers). So it's a toss-up whether you code I%↑2 or I%*I%, but I%↑3 is much tighter than I%*I%*I%.

Again, logical operators generate a fixed amount of code: 12 bytes, for 50 cycles. You might save a little in some cases by coding an appropriate addition or subtraction instead [1]; lower.case%=upper.case%+20h is five bytes and 13 cycles shorter than lower.case%=upper.case% OR 20h. This approach works only if the variables are to take on restricted sets of values, and so might require range checking that would swallow any savings.

Real work

Real-number operations are all in a fairly standard form. First of all, each variable is given an eight-byte allocation in the data area, with its address coded in line wherever it is used. Real constants ([2], p. 8) are referenced by pointer in the same way, but they are stored in what I call the constant area, below the executable code. (If that seems trivially obvious, bear in mind that integer constants are never stored separately; if needed explicitly at all, they are embedded in the code.)

A simple arithmetic assignment, such as R1=R2+R3, generates 15 bytes of code, for — oh, let's forget the cycles:

```
LD      DE,R2 pointer
LD      HL,R3 pointer
CALL    ?ARMM       Add Reals, Memory
                    + Memory
                    (result is eight bytes
                    on Stack)
LD      HL,R1       destination pointer
CALL    ?TRMS       Transfer Real to
                    Memory from Stack
```

For other operators, substitute library routines like ?SRMM, ?MRMM, and ?DRMM in place of ?ARMM. There are no pyrotechnics when the arguments are small constants. The logic is the same if R2 or R3 is a real constant, but costs eight bytes in the constant area.

If an argument is an integer constant, however, we invoke CB80's polite warning about mixed-mode arithmetic ([2], p. 13). But let's be a bit more specific and see just what the trade-offs are. The statement R1=R2+123 (or R1=R2+-I%) creates 18 bytes of code but saves eight in the constant area, for a gain of five bytes:

```
LD      HL,123      or LD HL,(I%)
CALL    ?CRSH       Convert HL to Real
                      on Stack
LD      HL,R2       pointer
CALL    ?ARSM       Add Reals, Stack +
                    Memory
                      (result on Stack)
LD      HL,R1       dest.pointer
CALL    ?TRMS       Transfer to (R1)
                      from Stack
```

In a parallel fashion, the statement R1=123.0 generates nine bytes of code and eight of data, whereas R1=123 (or R1=I%) again saves five bytes.

What do we get out of this? That real-number arithmetic needs space, that we're not going to gain speed with any tricks, and that mixed-mode expressions can save space but cost speed. Note the healthy incentive to dedicate real variables for frequently-used fixed values — encouraging good programming practice: every mention of a real constant costs another eight bytes in the constant area. To initialize them, use all the integer constants you can, saving another five bytes each. The twelve-byte overhead of TWO=2.0 will be amply recovered on the second use of TWO where 2.0 might have been. As for ZERO, each use is clear profit: CB80 will initialize it for you at run time.

REFERENCES

1. **Robert P. VanNatta,** *Lifelines,* March 1983
2. *CBASIC Compiler Language Reference Manual,* Third Edition, December 1982
3. Tom DeMarco, *Structured Analysis and System Specification,* Englewood Cliffs, NJ; Prentice-Hall, 1979

Feature

Learning C Inexpensively

by Van Court Hare

In Spring 1983, two inexpensive and popular C compilers were released in new revisions: Walt Bilofsky's C/80 (The Software Toolworks, 11478 Glorietta Drive, Sherman Oaks, CA 91423, $49.95, Version 2.0) and Leor Zolman's BDS C (BD Software, P.O. Box 9, Brighton, MA 02135; Lifeboat Associates, and others, $150, Version 1.5).

Although subsets of C are implemented in C/80 and BDS C, the compilation speed of these eight-bit, CP/M-80 2.2+ oriented products makes either suitable for learning the C computer language.

In addition, users of 16-bit (PCDOS, CP/M-86, MPM-86 and CCP/M-86) machines are truly blessed: now widely available, Mark DeSmet's "C Development Package" (C-Ware Corp., 1607 New Brunswick Ave., Sunnyvale, CA 94087, and other distributors, $100, Version 1.5) is a full C compiler at a price anyone can afford. The DeSmet package includes a simple text editor, too.

"Learning C Inexpensively — and Quickly" will appear in several parts. The first segments concentrate on examples which employ the BDS C compiler. Later, I will discuss the DeSmet 16-bit package as this C tutorial, designed for new learners, continues.

Why Learn C?

Before proceeding, some background on C's philosophy and current status is essential.

Portability is the reason C is popular — and why C is worth knowing.

Software developers confronted with multiple CPU and operating system types have turned to C as a high-level language to avoid rewriting machine code as the hardware or OS environments change.

C's concept is simple: It contains only a handful of essential constructs — and completely excludes from its kernel all the I/O manipulations (such as file handling, console entry and display, and data formatting) which are heavily machine or operating system dependent. To port to another CPU type, only the "core" operations need be coded at the machine level. A standard library, written in C's kernel notation, supports I/O and other commonly used functions.

Portability is the reason C is popular — and why C is worth knowing.

A new version of C for a new chip can be generated rapidly (by revision and recompilation of the standard library using the new kernel). In turn, this new C compiler can then recompile application software (written in standard C syntax), usually with little or no adjustment. This transformation process permits users to insure previous work against technological change, builds a wider customer base, and thereby cuts potential per user software cost.

C also has many other features which speed software development and debugging. Because C automatically handles local variables in functions, extended specialized libraries of permanently useful modules can be developed, and programs can be put together in "software chip" form, following current hardware practice. C can work at the bit level when needed, or at as high a level as the user desires, through use of larger modules.

Finally, C was developed by two individuals, and most of the microcomputer C compilers available are the product of individuals — not committees! The spirit of no compromise prevails. Most compiler writers have adhered closely or exactly to the specifications set out by Brian W. Kernighan and Dennis M. Ritchie in *The C Programming Language,* (Prentice-Hall, 1978) (usually abbreviated "K&R" in the literature). Appendix A of this text provides the complete reference manual for the language, and is the *de facto* standard for all users. Accordingly, C is, in reality, the most standardized computer language available today.

Both large and small software houses have clutched these advantages of C to their respective breasts in recent months, so many of your favorite packages are now written in C, or have been converted to C for the reasons noted. For example, DRI (CP/M's developer) has now adopted C as that firm's standard development language. Microsoft, MicroPro, and SuperSoft (among others) have taken the same tack.

So, for those who already know another language, you can see why C is definitely worth learning now.

Those with no prior programming experience should be warned, however, that C (as usually presented) demands that you know a wide range of programming concepts before you can easily get off the ground. That is, you should already know about loops, tests, subscripts, arrays, and so on. Prior experience with BASIC, FORTRAN, or Pascal is adequate. Those also familiar with a machine language certainly will have no trouble.

You must also realize that C is always compiled, and so does not have the easy diagnostics or development ease of an interpreted language, like BASIC. C assumes your source program is nearly perfect at compilation time, otherwise you will waste effort and get bizarre error messages from the compiler.

Editor's Note:
TINY C, a small subset of the language, is good for teaching structured programming and features an interpreter as well as a compiler.

Preparing to learn C

Some prior study of C's rules is sug-

gested before you attempt to prepare any programs on your own.

Go out and buy two books: The Kernighan and Ritchie text, previously mentioned, plus Alan R. Feuer's *The C Puzzle Book*, Prentice-Hall, 1982, $33 for the pair. Both are bound in the same style to be used as a set.

The index to K&R is more than complete; it references all key terms plus all examples mentioned in the text, many of which make up the standard C library. You will have no trouble finding what you need. K&R has a nice tutorial as Chapter 1, and Appendix A is the C Reference Manual.

Feuer's example quiz programs (with full output and explanation of method) often clarify otherwise dense points in K&R, and Feuer's appendices tabulate all C operations and their precedence as a handy reference. Alas, Feuer has no index. A right curly brace was omitted in error at the foot of page 53 of Feuer, so, write it in when you get the book.

One useful table is missing from both K&R and Feuer: A complete list of all functions in the standard C library. Such a table is included in: Edward M. Rifkin and Steve Williams, "The C Language: Key to Portability," *Computer Design*, August, 1983, pp. 143-150, with table at p. 148. (This issue contains a special section on 16-bit software.) Neither C/80 nor BDS C contain all of the functions shown, but the cited table is a good check list.

In addition, try to get the August, 1983 issue of *BYTE*, devoted entirely to C. Example programs abound. This issue contains technical comparisons of C compilers, and other information of general interest, including a tutorial (which continues to September). An extensive annotated C bibliography is included, should you wish further information.

Be sure to make a quick tour through this material — as well as the manual for your compiler. Such prior study will save lots of time later on.

Text editors

Creation of C programs requires the use of a text editor, which the user must have. The "N" option of WordStar will suffice, but getting in and out of WS is a slow process. Text editors especially designed for programmers, notably PMATE (Lifeboat) and VEDIT (most distributors), are ideal. Each of these highly regarded and popular editors has extensive macro substitution capability and its own programming language. Needless to say, CP/M's ED is a cumbersome last resort.

Which eight-bit C Compiler?

For CP/M-80 users who wish to avoid machine level detail, BDS C is the best bet. C/80 requires more typing and greater knowledge to get going.

Ideally, completeness and compilation speed are the criteria for a "learning" compiler, not necessarily compact object code or execution speed. Unfortunately, complete, inexpensive, and fast seldom come in the same package. Here is a brief comparison of C/80 and BDS C. (Other eight-bit compilers were not considered for this article for reasons outlined in Christopher Kern's review of eight-bit C compilers, *BYTE*, August, 1983, p. 110.)

1. **Completeness.** The elimination of floating point and long integer arithmetic from both BDS C and C/80 (inherent memory and speed limitations in eight-bit machines) will not permit users to run all of K&R's Chapter 1 tutorial programs. This is a pity, and will cause some confusion at the outset. However, examples requiring *float* and *long* variables may be skipped at first without too much loss. Stick to integer arithmetic. (As noted later, BDS C **does** support *float* and *long* by use of library functions, but the detailed code does not follow K&R exactly. C/80 has **no** *float* or *long* support.)

Good documentation of your first C compiler is essential, and both C/80 and BDS C pass this test admirably.

Two serious additional omissions from C/80 seriously affect swift learning: C/80 does not support arguments in preprocessor *#define* statements (as used extensively in both K&R and Feuer), and the high-level formatted data input function *scanf( )* as well as the high level formatted file input function *fscanf( )* are both omitted from the C/80 library. The latter omissions force you to excessive detail at too early a stage, which will discourage the novice. Near conformity of your compiler to the existing textbook examples is mandatory if you want to learn C quickly. (BDS C *does* support *#define* arguments, and both *scanf( )* and *fscanf( )* are in its standard library.)

2. **Documentation.** Good documentation of your first C compiler is essential, and both C/80 and BDS C pass this test admirably.

C/80's 35 page manual may appear brief at first glance, but it is the most carefully prepared 35 pages you may ever see, beautifully organized, tight, well indexed, and downright pretty. This brevity may be an asset to the novice. I found that a comparison of the C/80 manual to the BDS C manual was an instructive exercise. Because the price of C/80 is so low, you might consider the purchase of both C/80 and BDS C — just for the sake of comparison.

The BDS C 181 page manual has been completely rewritten for Version 1.5 (using Mark of the Unicorn's Final-Word text processor — which was in fact written in BDS C!). Offset masters were typed on a Diablo printer using a "96-Bold-PS" metal daisy-wheel. The result is an exemplary manual — as well as a demonstration of BDS C's practical usefulness. Leor Zolman has also provided a number of personal tutorial chapters, observations on common beginner errors, and similar supplementary material. Overall, the learner will like this well indexed manual for its completeness and clarity.

3. **The C Library**. The immediate usefulness of a C compiler lies in the standard library's contents. Neither C/80 nor BDS C provide the complete C library (you must go to a much higher cost 16-bit compiler like Lattice or DRI C for that).

C/80 provides only the essential I/O functions, but does support both sequential and random access disk files, and the standard console functions. The standard conversion func-

tions, such as *atoi( )*, do not appear, but you can copy them from K&R. C/80's brief library at least shows you what essential is! You can add on what you want — if you have the time. Nevertheless, the brevity of C/80's library is its greatest shortcoming.

BDS C, on the other hand, goes to the other extreme and provides most of the standard library plus a number of useful special functions: delays, random number generators, and so on. All of the integer to character, upper to lower case, and similar conversion functions (usually directly from K&R) appear. Each function is detailed in the documentation, although the manual would be better with an example for each library entry.

BDS C does support *float* and *long* as separate, non-standard library functions (not as standard C code); some setup is required, however, as described in the manual.

In short, for initial learning followed by productive use, the BDS C library is a pleasure, and eliminates much typing and fiddling.

Both C/80 and BDS C use the random access file functions of CP/M 2.2 (or later), which won't work with earlier versions. The user should also note that CP/M specific library functions (such as bdos( ) calls) are operating system specific.

4. **Speed and Flexibility.** C/80 compiles swiftly to an intermediate .ASM (assembly language) file, which yields machine code quickly using the absolute (no linking) assembly program provided. The resulting object code is very compact. Interestingly, C/80 is written in C/80 itself.

Because separately linkable modules are desirable for flexibility, C/80 also provides an alternate linkable output option — but uses Microsoft Macro-80 (M80/L80, not supplied) for this purpose. Macro-80's Spartan documentation and extra steps add a whole new learning burden, and increase effective "trial" time. The Macro-80 package alone exceeds BDS C's cost. With BDS C, you get a special linker free.

Readers not familiar with linking loaders may like Michael Olfe's "The Missing Linker," a tutorial, (*PC Magazine*, September, 1983, pp. 395-402).

BDS C is noted for its compilation speed. The compiler is written in 8080 machine code, and compilation of an entire program is done in the computer's high speed working memory. Compiler output is in the form of special .CRL (not .REL) relocatable intermediate modules, which may be separately compiled. (An .ASM file is not produced.) These .CRL program segments can be linked — using BDS C's linker, which *is* supplied — to produce programs larger than memory limits. Overlays are supported. The extensive library, also partially written in machine code for speed, produces an overall package which is a delight to use.

To illustrate, BDS C compiles, and links, Example 1 (to follow) in about 45 seconds on a 4 MHz. Z80 machine.

Only two simple commands are needed to compile and link to executable code. (Version 1.5 of BDS C is 20 percent faster and produces object code which is 25 percent more compact than prior versions.)

5. **Other features.** Both C/80 and BDS C have added features worth noting.

A number of sample C programs come on the C/80 distribution disk in source code form, and these are useful in testing C/80 and illustrating C. C/80 also provides a unique run-time profile utility which will show the user which parts of a C program are most heavily used, so those segments may be fine-tuned for speed, if desired.

In addition to the *float* and *long* library supplements already mentioned, the BDS C distribution disk contains an alternate linker, librarian, debugger, various definition files, and a number of simple and complex example C programs and utilities, most of which are documented in the manual (or on the disk). Many of these items are new version 1.5 features; the debugger and its corresponding compile options are an example.

TELEDIT, a complete communication and file transfer program which uses the MODEM7 protocol, is one of the sample C source programs, and most distributors include the game "Othello" in C source, too.

BDS C distribution disks vary somewhat by vendor, but anything missing — plus more — can be had from the BDS C User's Group (Box 287, 112 N. Main St., Yates Center, KS 66683, $10 membership), which now supports C generally, and distributes public domain C utilities and programs for a minimal duplication fee — testimony to BDS C's maturity.

6. **Conclusion.** Although C/80 is a satisfactory learning tool for those on a limited budget, BDS C — even at the higher price — is a superior buy for its: (a) total compilation-link speed and ease, (b) extended library, (c) special added features, (d) extensive manual, and (e) user support group.

The examples shown later in this article are (until noted otherwise) prepared with the BDS C on an Intertec Superbrain I QD machine. Comments on C/80 appear when appropriate. Now consider an example to exercise your C compiler.

C provides a powerful macro preprocessor so machine dependent or cumbersome text strings can be defined in an easy to change header.

The structure of C

The texts and manuals cited at the beginning of this article explain the nitty-gritty of C adequately. Slighted, however, is a firm insistence that users get the overall structure of a C program in mind — before they write code.

Example 1 was designed to show how a C program must be put together. The objective of this program is to draw a border around the CRT terminal screen through use of two generalized graphics functions. The method is first to fill the screen with "*" 's, then fill a slightly smaller rectangular box with spaces. The visual effect is that of an upside-down windowshade.

Before you attempt to compile and run Example 1, please read to the end of this article: Example 1 may require slight modification for your terminal.

1. **The Total C Program.** A C program is prepared as a set of functions, one (and only one) of which must have the name *main( )*. A set of parentheses, which may or may not contain arguments, must *always* be included after a function name. In Example 1, the functions used are *main( )*, *boxfill( )*, and *linehr( )*. Function *main( )* has no arguments in this example, because none are passed to it.

2. **Preprocessor Definitions.** C provides a powerful macro preprocessor so machine dependent or cumbersome text strings can be defined in an easy to change header. Before compilation, the preprocessor substitutes for each instance of the macro name its definition, an action comparable to the search-and-replace action of a wordprocessor. In addition, arguments of defined macros are correctly processed (in standard and BDS C, *not* C/80). For example, for A(i,C) of boxfill( ) in Example 1, the preprocessor will substitute *i* in the definition in place of *R*. The preprocessor information may include reference to entire files of previously defined macros (*#include*), as illustrated in both K&R and Feuer.

Before you attempt to compile Example 1, replace the *#define* definitions shown with those appropriate for your computer's CRT. Use your text editor to type Example 1 and make the necessary changes; then save the text file as EX1.C — the extension .C is required by the compiler.

Specifically, your CRT's reference manual will show you the code for clearing the screen (shown as ERASE) and for direct-cursor addressing (shown as AT(R,C)). The Superbrain I uses ASCII character 12 (Form Feed) to clear its screen, leading to the definition ERASE putchar(12), as shown. One space delimits ERASE and its definition. Similarly, the Superbrain I definition for AT(R,C) is given. Use the possibly different escape sequence for your machine. The string "\033Y" shown is C's equivalent of ESC-Y. See the preprocessor notes in K&R at p. 86 and 207 and in Feuer at p. 23 and 69.

In Example 1, macros which appear anywhere in the program have been set in upper case letters. This distinction is good typing style, because it permits you to see where macro substitution will take place.

3. **Subfunctions.** The remainder of a C program consists of an appropriate collection of functions. All function names mentioned in the program must (a) exist somewhere within the total program, (b) appear in the standard library, or (c) be linked from specialized collections of prior user-defined functions. In Example 1, functions such as *putchar( )* are picked up from the standard library automatically; those such as boxfill( ) and linehr( ) are supplied within the program. No prior user-defined functions are employed in Example 1.

4. **Variable declaration.** As shown in boxfill( ) and linehr( ), all variables in the argument list (shown between parentheses following the function name) must be declared immediately after the function name — and before the first left curly brace ({). For example, boxfill(R,C,width,height,ch) requires the declaration of five variables, as shown in Example 1. (A semicolon must terminate each declared variable list. You delimit items in the declaration list with commas.)

The body of the function is set off by left and right curly braces ({ }), but before any internal C code can occur you must also declare the type of any additional variables that will be used. (See the variable *i*, which is used in boxfill( ) and linehr( ), and note how it is declared and where.)

5. **Variable Scope.** Unless explicitly stated otherwise (by methods not shown here), the scope of variables used in a given function will be "local" — or in C terminology "automatic."

Specifically, when a function is called, the values of its arguments (not their addresses) are passed to it. A temporary, private copy of all argument values is made for the immediate use of the function. Internal function variables are also local. When a function terminates or returns, all its local variables are destroyed, so no confusion between modules can occur. A *return( )*, if used, normally passes back a single integer or character. Special treatment is required to return strings and multiple numeric values. See K&R pp. 65-71.

No *return( )* statement is shown in Example 1, because the graphics routines do not need to return a value; they take direct output action via *putchar( )*. In Example 1, each function terminates when its last right curly brace (}) is encountered.

C functions are made independent of each other by the "automatic variable" process just described, and variables of the same name can be used in various functions in a given program. The fact that boxfill( ), Example 1, uses *i* as a utility counter, then calls linehr( ) — which also uses *i* as a counter — causes no problem. In this way, through creation of independent modules, you can build up a library of your own software tools to use in applications in which you specialize.

When a function terminates or returns, all its local variables are destroyed, so no confusion between modules can occur.

6. **How a C Program RUNs.** The *main( )* program of Example 1 is rudimentary, but it controls the action of the other functions in this C program. In this case, the calling arguments are literal values, and *main( )* has no arguments. Consequently, no variables need be declared in *main( )*. As usual, the statements in *main( )* are processed from the top, one at a time. In C, the mention of a function name (with specified parameters, or none, as appropriate to the function) is equivalent to a function (or "procedure") call. In particular, note in *main( )* that the named functions show literal arguments, and that these specifications agree left-to-right in count and type with those arguments required by the called function. The unquoted numeric literals of *main( )*'s calling functions are taken as integers by the called function; whereas, the single quotes (in '*' or ' ' respectively as shown) identify type

►

char. Such correct type matching of calling and called arguments is essential.

By now, you should be able to trace the action of *main*( ) through the action of boxfill( ) and linehr( ).

To compile and run Example 1, three steps are required (for the BDS C system): (a) with the compiler on disk A: and the source code for EX1.C on disk B:, type the command CC B:EX1 followed by a RETURN; (b) when compilation is complete, type CLINK B:EX1 and a RETURN; (c) when the link process finishes, type B:EX1 and you will see the graphic results. (Lower case commands and file names may be used, if you like; CP/M converts lower to upper case automatically.)

If your typed program fails to compile and you get cryptic error messages, check for a missing semi-colon or curly brace in your text. Forgetful omission of a semi-colon or a curly brace is *the* most frequent first error of new C users, and such omissions will cause your compiler to generate multiple, uninformative diagnostic error messages!

To test your understanding of Example 1 and to get some practice with your compiler, modify the literal values in *main*( )'s call arguments by use of your text editor, then recompile, run the program, and see what happens. For example, try to draw a small box in the middle of the screen; then as a second exercise, try to draw three small boxes at different screen positions which look like three overlapping cards. Be careful in your choice of calling values; for simplicity, no error checking is done in boxfill( ) or linehr( ). Inappropriate R, C, width, and height values will draw off the screen.

The next installment

The high-level I/O functions in C — starting with *printf*( ) and *scanf*( ) — permit rapid program development and are the subject of the next installment. Take a look also at the standard library I/O functions *putchar*( ), *getchar*( ), and *puts*( ). Some knowledge of pointers will be required, so check out K&R pp. 89-103.

```
/* EX1.C    EXAMPLE 1: A GRAPHICS EXAMPLE TO ILLUSTRATE C'S STRUCTURE */

/* Header — Specify some terminal details: This is for
Intertec Superbrain I QD. Redefine for your terminal. */

#define ERASE putchar(12)                                   /* clear screen */
#define AT(R,C) puts("\033Y");putchar(R+31);putchar(C+31)
                                                    /* for direct cursor addressing
                                                    with origin at 1,1. For origin 0,0
                                                    replace 31 with 32, above. */

/* setup and control desired box size and placement */

main( )

{
                    ERASE;
                    boxfill(2,1,80,21,'*');         /* use single quote for ch */
                    boxfill(3,2,78,19,' ');
                    AT(23,1);
                    puts('EXAMPLE 1, DISPLAY
                    BOX');
}

/* fill rectangle of size height × width with ch; NW corner at R,C */

boxfill(R,C,width,height,ch)

                    int R,C,width,height;           /* declare argument types */
                    char ch;
{
                    int i;                          /* declare internal variable */

                    for (i=R; i<=(R+height-1); ++i)
                    {AT(i,C);
                    linehr(width,ch);}              /* draw line */
                                                    /* curly braces define
                                                    multi-statement "for" loop */
}

/* draw a line of size width using ch to right from cursor */

linehr(width,ch)

                    int width;
                    char ch;
{
                    int i;

                    for (i=1; i<=width; ++i)
                        putchar(ch);
}
```

Feature

New Products Ideas And Trends In Digital Hardware

by Jon Wettingfeld

New electronic mail system launched nationwide by MCI

MCI Communications Corp. is now staking its claim in the burgeoning "time sensitive mail" market, one currently worth eight billion dollars a year. Called MCI Mail, the telecommunication-based system is designed to reach anyone anywhere in the continental United States faster, and at a price that is 90% lower than competitive time sensitive mail systems.

MCI offers four types of service, each with a different speed and price. These are: 1) *instant mail*—delivery from terminal to terminal via MCI's electronic mailbox (i.e. the text is held in storage until addressee retrieves it); 2) *four-hour mail*—with paper copy delivered by courier based in the metropolitan areas of fifteen major cities within four hours, regardless of point of origin of the message; 3 *MCI mail overnight letter*—electronically transmitted messages hand-delivered by noon of the following day; and 4) *the MCI letter*—mail transmitted electronically to the MCI postal center nearest its destination, then brought to the nearest U.S Postal Service for regular delivery.

Like Western Union's "Access" computer mail system, MCI makes substantial claims about having overcome protocol incompatibilities among electronic terminal devices. Thus, it should be usable with "virtually any personal computer, word processor, electronic typewriter, data terminal, telex or other digital communications device." With nearly any combination of devices compatible, messages sent on (e.g.,) Xerox or other word processor can be read in turn by a Wang word processor, or vice versa.

Unlike the U.S. Postal Service's currently operational E-Com electronic mail system, no minimum number of messages is required, and registration is offered at no cost to anyone with a digital device. Like E-Com, the service is accessed by a local phone call. The call connects to one of the fifteen special MCI mail centers. Over longer distances, messages are routed through a toll-free number. All messages originate at the subscriber terminal. Those delivered electronically are stored in an "electronic mailbox," then transmitted to recipients' terminals. Of course, mail addressed to parties without terminals can be routed as paper-based mail, as already noted.

There are several innovative firsts in the system. One is the hard copy mail distribution system contracted to Hewlett Packard. This utilizes HP 3000s with Hewlett Packard 2680 laser printers. The printers, along with graphics reproduction facilities, can be used for reproducing letterheads, forms and signatures. This puts the network technologically ahead of the Post Office's E-Com system, which has scheduled the use of such printers to begin in 1984. The word processing translation facilities, so important to the success of such an electronically pluralistic mail network, have also been newly designed by Racal Telesystems. Other major contributors involved with the system's development include Digital Equipment Corp., which developed the custom interfacing and mail switching technology and implemented the software for the system on Vax 780s.

Sidereal Corporation introduces new line of high- and low-end office hardware

Sidereal Corporation originated, and is a leading manufacturer of, intelligent multifunction, multiport message communications terminals. Now Sidereal has introduced three new products which deal in turn with: lowering costs in store and forward message switching, integrating powerful word and data processing with state of the art communications, and lowering costs of a basic electronic message terminal while at the same time retaining many switching capabilities present in "high-end" message communications systems. The Micronet Message Switch is a compact desktop store and forward system that accommodates all popular communications protocols. Microprocessor-based, it is said to use current technology to lower costs of message switching functions. It features speed, code and protocol conversion. It comes either as a complete software/hardware system, or unbundled, with software modules added as desired.

The Micronet 12 Message Terminal is a single-line terminal incorporating high-end features of more expensive multiport systems. It is designed to replace conventional mechanical or electromechanical telex terminals, and is said to be simpler both in use and design. Thus, it eliminates much of the extensive training time usually needed to learn to run such terminals. User friendliness is increased by the fact that the unit provides on screen help to operators preparing messages for transmission. In addition the terminal is highly versatile, being usable for electronic mail or data entry, or as a public or private database retrieval terminal. It is, therefore, ideal for use in a corporate network or as a free-standing multifunction CRT.

The most powerful new Sidereal product is the Micronet R85 Multi Workstation Office Information System. With a user-friendly operator's environment employing pictorial symbols ("Icons") and six global commands to facilitate most operations with only two keystrokes, the system allows the operator to readily perform most file maintenance and communication functions. The system contains a Unix-like operating system, an integrated spreadsheet, business graphics and advanced word processing capabilities, and does filing and electronic mail. The operating system itself allows the use of a broad range of applications programs written for the Unix system.

Since it has extensive communications software, the Micronet R85 is readily interfacable with specialized communications networks. With its sophisticated communications capabilities, the system is described as being "virtually unlimited in the size and geographic scope" of the networks in which it can be linked.

Small bytes

Moving from the grand level of international communications to the more modest realm of home entertainment and the low end micro, Rana systems has introduced the Rana 1000 intelligent disk drive, showing that it isn't just "playing games" with the Atari computer. Described as the first of its kind, the sleek, low profile peripheral measures two-thirds the size of the standard Atari drive. It is compatible with all Atari systems modes and software. Features include error detecting and indication for 21 different conditions. These are signaled by two-character LED readout and indicator lights. Additionally, five touch-sensitive electronic function keys are displayed on the front panel. Indicators allow the drive to denote single or double density operation, give track identification, and show which track is being written upon or read.

Altos Computer Corporation reports the use of a novel interface with their equipment. Remington Products has significantly improved productivity levels by using an Altos Multi user system with three Micropad data entry terminals. Clerks hand-write sales tickets and this simultaneously enters data into the system. This data, in turn, is collected nightly at the company's regional headquarters from 29 Remington stores by polling the in-store computers via modem. This system has eliminated errors and delays in data transfer, and reduced the amount of time needed to prepare a sales ticket.

Product Status Report

New Products

TALISMAN

Disco-Tech
600 B Street
PO Box 1659
Santa Rosa, CA 95402
(707) 523-1600

This program is designed for terminal translation and multiple keyboard redefinition. TALISMAN makes it possible for any microcomputer terminal to emulate any other terminal, including DEC's VT-100. It occupies a max. 3K RAM and lets any CP/M 2.2 software run on any terminal, without regard for the terminal for which the software was originally written or configured. TALISMAN enables any CP/M microcomputer to emulate the screen control features of any micro, mini, or mainframe terminal. It does not support block mode data entry or editing. TALISMAN also lets its users reprogram any keyboard completely, creating as many as 255 overlays, which can be saved and recalled at any time for programming, wordprocessing, or special applications. It is user friendly and has a menu-driven configuration program.

Requirements: CP/M 2.2
Price: $125

G&G 1040

G&G Software, Inc.
610 Park Blvd.
Austin, TX 78751
(512) 458-5760

This tax preparation program is designed to make use of IRS supplied forms as input sheets and to leave calculations and printing for a microcomputer. Input is to screens which match IRS forms. The program calculates an entire return in 10-15 seconds. It makes all choices for the least tax consequences. It calculates everything possible including optional state sales tax, excess FICA, earned income credit, income averaging, minimum tax, alternative minimum tax and underestimate penalties. The Professional Series processes 39 IRS schedules and includes a batch computer/print mode, and a depreciation module. The Starter Series processes 15 IRS schedules but does not include batch operations nor a depreciation module.

Requirements: CP/M-80 or CP/M-86
Price: Professional Series —
$750 (annual updates $250)
Starter Series —
$195 (annual updates $125)

G&G GL

G&G Software
(for address see above)

This program is for use in an accounting practice which supports record-keeping for its clients. It can also be used for other businesses where several sets of books are being kept. Its features include: several input methods are supported all of which require entries to be in balance, payroll records of employees are updated at the same time entries are made to the general ledger, and account numbers for the general ledger accounts are assigned as the user elects up to 99999.9. There is no limit on the number of accounts or the number of transactions except for the disk space that is available in the user's system. All output is formatted for 8½" × 11" paper. In addition to the usual reports, Statements of Changes in Financial Position are prepared. Departmentalization and budgeting are supported. All payroll reports are prepared as needed. A complete Fixed Asset or Depreciation module is included which calculates depreciation for all usual methods including ACRS.

Requirements: CP/M-80, CP/M-86, 64K
Price: $495

SECURITY

the ANSWER in COMPUTERS
6035 University Ave., Suite No. 7
San Diego, CA 92115
(619) 287-0795

SECURITY provides password protection for all programs without impacting normal processing and with no modifications to the operating system. SECURITY-PLUS allows the additional option of password protecting individual files. SECURITY-PLUS-LOG maintains a complete record of program and file activities.

Requirements: CP/M-80 2.2, CP/M PLUS or MP/M-80
Price: SECURITY — $50

SECURITY-PLUS — $100
SECURITY PLUS-LOG — $150

Oper 8

Selkirk Computing Systems
17131 Hofer Court
Lake Oswego, OR 97034
(503) 241-8448

This program provides operations management with systems and procedures necessary for efficient and accurate purchasing, sales order and inventory management. It includes three systems: Purchasing, Sales Order Control and Integrated Inventory Management, which includes a Bill of Materials processor.

Requirements: dBASE II, CP/M, PC-DOS or MS-DOS
Price: Purchasing — $500
Sales Order Control — $750
Integrated Inventory Management — $2250

PROGRAM MAP

The Software Store
706 Chippewa Square
Marquette, MI 49855
(906) 228-7622

This program is a cross reference tool for Microsoft BASIC programs. It speeds program development and helps accurately "debug," modify, convert and document BASIC programs in a minimum amount of time. PROGRAM MAP produces alphabetical lists of variables, commands, functions, constants, quoted strings and line numbers. Each "word" is listed with the line number(s) in which it is found. Print width and paging control is included. It is fast and easy to use and runs directly under CP/M.

Requirements: 8080, 8085, Z80, 48K, CP/M
Price: $150

New Books

WORD PROCESSING FOR SMALL BUSINESSES
by Steven F. Jong

Howard W. Sams & Co., Inc.
4300 W. 62nd Street
Indianapolis, IN 46268
(317) 298-5400

This book describes the elements of word processing systems, discusses the decisions that need to be made, explains the options and offers sound advice. Descriptions cover more than 50 hardware and software products for dedicated and nonspecialized systems alike, down to disk drives and printers. Also included are performance check lists, a sample service contract, charts for comparisons of similar products and forecasts of what's ahead in microcomputing.

Price: $11.95

WHAT *DO* YOU DO AFTER YOU PLUG IT IN?
by William Barden, Jr.

Howard W. Sams & Co., Inc.
(for address see above)

This book presents a complete tutorial covering use of microcomputer hardware, software, languages, operating systems, and data communications, followed by a second tutorial on workable solutions to the practical problems that occur during their use. The text also talks about hardware and peripherals, plus advanced technology in such devices as print heads, bubble memory and high-resolution graphics. Barden covers packaged applications software, operating systems, programming languages and other software topics, including a checklist for the software buyer.

Price: $10.95

1984 PROGRAMMER'S MARKET

Writers Digest Books
9933 Alliance Road
Cincinnati, OH 45242
1-800-543-4644 (credit card orders)
(513) 984-0717

This book is aimed at microcomputer programmers. It gives information on markets for their software. It lists over 500 software publishers, arcade game publishers, magazines with information on whom to contact and where, submission requirements, royalty/payment terms, available contract work, and tips from buyers on selling software. In addition marketing techniques are discussed such as: preparing a query letter, proposal package, copyrighting programs, writing user manuals to accompany software, documenting programs, writing user friendly software, and writing best-selling game programs.

Price: $16.95

New Versions

NEW VERSIONS

PANEL-PC and -86 (for LATTICE C)	v.5.0
COBOL-80	v.4.66
GrafTalk	v.2.15
CP/M WorkShop	v.2.0
Precision BASIC for MS-DOS	v.1.7
Precision BASIC for CP/M-80 (Z80)	v.1.2
FORMULA-II	v.2.2
MagicPrint	v.2.0
PL/I-80	1.4
HALO	v.1.33
w/HERCULES board	v.1.85
MAG/base-1,2,3 compiled under CB80	v.3.2
SMARTKEY-II (CP/M-80 and MS-DOS)	v.1.0
SMARTPRINT	v.1.0

REACH 2.0 Model and File Transfer

The Software Toolworks
15233 Ventura Boulevard, Suite 1118
Sherman Oaks, California 91403
(213) 986-4885

The REACH 2.0 Modem and File Transfer program is capable of automatic programmed interactions with remote systems.

With REACH 2.0, you can dial a remote system (using an autodial modem such as the Hayes Smartmodem), log in automatically, even read your mail, send messages, and log off, all without operator intervention. This is accomplished by executing command files, written in a simple programming language, which can choose actions based on strings sent from the remote computer. Sample Micronet login and mail reading command files are included.

REACH 2.0 is menu configurable for easy setting of baud rates, serial interface parameters and file modes. It retains the easy-to-use "smart terminal" features of earlier versions.

Registered owners; update $10 plus shipping.

Lifelines™/ **The Software Magazine**™
1651 Third Ave., New York, New York 10028

Second Class Postage P
At New York, N.Y.